APPLYING INTER-LAYER CONFLICT RESOLUTION TO HYBRID ROBOT CONTROL ARCHITECTURES

A Dissertation
Presented to
The Academic Faculty

by

Matthew D. Powers

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing
College of Computing

Georgia Institute of Technology
May 2010

APPLYING INTER-LAYER CONFLICT RESOLUTION TO HYBRID ROBOT CONTROL ARCHITECTURES

Approved by:

Dr. Tucker Balch, Advisor
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Dr. Ronald Arkin
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Dr. Henrik Christensen
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Dr. Magnus Egerstedt
Department of Electrical and
Computer Engineering
Georgia Institute of Technology

Dr. Anthony Stentz
Robotics Institute
School of Computer Science
Carnegie Mellon University

Date Approved: January 14 2010

To my wife, Tzu-Wei, without whom, the last seven years would have seemed much longer.

To my parents, who taught me to show gratitude for my God-given talents by using them to my utmost ability.

To God, in gratitude.

ACKNOWLEDGEMENTS

As I found myself in Atlanta in August of 2002, I had no idea the great fortune that was in store for me. Within a few days, I was in touch with Tucker Balch, a new professor who, in time, would teach me everything I needed to know about robotics research. Within a few months, I was working on some of the most interesting research I could imagine. Within a few years, I was marrying my beautiful wife. I can only thank God for his providence.

Successful robotics projects of any appreciable size are inevitably the result of many people working together. I have to thank Yoichiro Endo and Alex Stoychev for teaching me the ropes early in my career, specifically, that good execution is just as important as a good idea. Many others contributed to the research presented in this document: On the LAGR project, Dave Wooden (whose ideas and implementations proved to be invaluable to this work), Jie Sun, Dongshin Kim, Ananth Ranganathan, and Charles Pippen. On the Urban Challenge, Dave Wooden (who, again, had much to do with making the platforms in this thesis work), Ben Johnson, Bob Schafrik, Karl Kluge, Vince Camp, and Tom Collins. On the RCI project, Dennis Ding (for carrying the project while I was finishing my thesis).

I certainly have to thank the faculty who got me where I am now. Jim Rehg, Frank Dellaert, Charles Isbell and Ron Arkin for teaching me the fundamentals of perception, planning, control and learning. Henrik Christensen for having the confidence in me to position me as a main contributor to our Urban Challenge effort. Magnus Egerstedt, for serving as a second advisor and constantly nudging me out of the nest. Tucker Balch, for being my advisor and teaching me how to deliver the goods.

All work and no play would have made me a dull boy. I have to thank Steve Marlowe, Bob Amar and Keith O'Hara for good drinks, lively discussion and a healthy dose of nonsense. As a "starving" grad student, I was rarely hungry, and I have my wife's family to thank for a good meal and a good laugh on a regular basis. I also have to thank my own family for their financial, moral and spiritual support over the years. Finally, I must thank my wife, Tzu-Wei, whose support, advice and love have made this process that much easier.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this document, we propose and examine the novel use of a learning mechanism between the reactive and deliberative layers of a hybrid robot control architecture. Balancing the need to achieve complex goals and meet real-time constraints, many modern mobile robot navigation control systems make use of a hybrid deliberative-reactive architecture. In this paradigm, a high-level deliberative layer plans routes or actions toward a known goal, based on accumulated world knowledge. A low-level reactive layer selects motor commands based on current sensor data and the deliberative layer's plan. The desired system-level effect of this architecture is that the robot is able to combine complex reasoning toward global objectives with quick reaction to local constraints.

Implicit in this type of architecture, is the assumption that both layers are using the same model of the robot's capabilities and constraints. It may happen, for example, due to differences in representation of the robot's kinematic constraints, that the deliberative layer creates a plan that the reactive layer cannot follow. This sort of conflict may cause a degradation in system-level performance, if not complete navigational deadlock. Traditionally, it has been the task of the robot designer to ensure that the layers operate in a compatible manner. However, this is a complex, empirical task.

Working to improve system-level performance and navigational robustness, we propose introducing a learning mechanism between the reactive layer and the deliberative layer, allowing the deliberative layer to learn a model of the reactive layer's execution of its plans. First, we focus on detecting this inter-layer conflict, and acting

based on a corrected model. This is demonstrated on a physical robotic platform in an unstructured outdoor environment. Next, we focus on learning a model to predict instances of inter-layer conflict, and planning to act with respect to this model. This is demonstrated using supervised learning in a physics-based simulation environment. Results and algorithms are presented.

CHAPTER I

INTRODUCTION

1.1 Motivation

Is it better to act with respect to short-term benefits, or long-term rewards? Are the two mutually exclusive? Can a balance be struck? Or, better yet, can one act to achieve both? While people in general have been dealing with these questions for millennia, robot software architects have been struggling with these questions, albeit in a different context, for decades. This work proposes a mechanism for rectifying a robot's actions toward long-term goals with its short-term constraints to improve external system-level performance.

Early proponents of deliberative approaches to robotics [36] championed a planning orientated approach. In this approach, models of the world and the effects of future actions are reasoned over, plans are generated, and actions are executed, all with an eye on optimality and predictability. In contrast, early proponents of reactive approaches [11] promoted a local approach. In this approach, the robot's sensory input is tightly coupled with its motor control. This approach values reactivity, robustness, and survivability over optimality and predictability.

Of course, no issue is black and white, and many modern systems use an implementation of a hybrid approach to robot control architecture. Hybrid approaches make use of decoupled layers of functionality to satisfy both the robot's immediate constraints and its longer-term objectives. Often, a *deliberative layer* provides a global plan as input to a *reactive layer*. The reactive layer then executes this global plan within its local context and according to its local constraints. For example, in the case of robot navigation in the area of field robotics, many current architectures

make use of a global deliberative planner and a local reactive controller [1], [49], [52].

The compromise between global objectives and local constraints is not always easy, and often the tradeoffs have to be empirically "fine-tuned" by the robot software designer. Either the deliberative layer's model of the world and the robot must be modified, or the reactive layer's interpretation of the deliberative layer's input must be adjusted. This process can be time-consuming and is subject to human interpretation of the robot's performance. It can also simply be difficult for a human to make sense of how all the degrees of freedom that a complex software system may contain might affect the robot's system-level performance.

Additionally, it may be desirable to integrate deliberative and reactive modules without explicit parameterization in the target environment. For example, one might want to use a path planner that uses overhead photography as a map on an unmanned ground vehicle. While the planner may have a relatively good model of the vehicle's reactive capabilities, given the variable nature of the imagery it uses, it may not be able to accurately predict the vehicle's performance in terrain it has not experienced before. If the planner were able to improve its model of the vehicle's performance in terrain as it experiences it, the a priori knowledge provided by the overhead imagery can be utilized more effectively.

As the tasks and environments we assign to robotic navigation systems get increasingly complex, the role of the robot software designer will have to shift from designing specific implementations, to designing general devices that are able to adapt and learn as new tasks and environments are presented to them. In this work, we present a formal framework for implementing learning between the layers of hybrid deliberative-reactive robot control architectures for spatial navigation. In this framework, system-level performance metrics are used as a supervisory signal to improve the interaction between the deliberative and reactive layers.

1.2 Research Question

Given the examples in the above discussion, it is desirable to have method for the robot designer to design all layers in a reasoned, disciplined manner, and then allow the robot to deal with any conflicts that may subsequently occur between the layers. This work proposes to improve the relationship between deliberative and reactive layers in a hybrid control architecture, in the context of spatial navigation, by defining a comparison function between the deliberative layer's model of the effects of its plans and the outcome of the reactive layer's interpretation of those plans. It also proposes the implementation of a correction mechanism, allowing the deliberative layer to modify its model of the reactive execution of its plans, based on a defined comparison signal. This leads to a formal research question:

Research Question - In the context of spatial navigation, and the scope of the interface of the deliberative and reactive layers, how can conflict between the models of the capabilities of a robot used in the layers of a hybrid deliberative-reactive robot control architecture be detected and corrected to improve overall system-level performance?

On the way to answering that question, we will examine two sub-questions:

Sub-Question 1 - How can conflicts between the deliberative and reactive layers' kinematic and dynamic models of the robot's relationship to the world be detected?

Sub-Question 2 - How can the robot learn from these detected conflicts to change its model of its relationship to to the world, in particular within the deliberative layer?

1.3 Contributions

In working to answer the research questions, several contributions are expected:

- Definition, within the context of a hybrid control architecture, of inter-layer conflict, produced by the conflict between a deliberative layer's model of the effects of its commands and the outcome of a reactive layer's interpretation of those commands.

- Method to modify the deliberative layer's execution based on learned models of inter-layer conflict to improve system-level performance.

- Method to modify the deliberative layer's execution based on incrementally learned models of inter-layer conflict to improve system-level performance

- Quantitative and qualitative experimental results demonstrating the value of the above contributions in the context of robotic spatial navigation.

1.4 Overview

Based around the afore-defined research question, this dissertation follows a logical progression. First, a background is established through a survey of related work. An approach to the research question is then developed, first generally, then formally. Finally, implementations of the approach are demonstrated on two different platforms.

A brief overview of each chapter follows:

Chapter 2: Background and Related Work - A survey of related work is reviewed. A background is established by reviewing work in behavior-based robotics, spatial planning and configuration spaces, and hybrid control architectures. Work related to the research question is also reviewed, including hierarchically consistent control systems, and learning between layers of hybrid control systems.

Chapter 3: Hybrid Control Architectures and Inter-Layer Conflict - A general approach to correction between control layers is explored. Qualitative

examples of conflict between layers of a hybrid control architecture are presented. Then, a qualitative approach is explained.

Chapter 4: Representation - The general approach explained in Chapter 3 is built upon, into a formal framework for learning between the layers of a hybrid control architecture. Hybrid control theory is used as representational framework for describing learning between layers as an optimization over an externally measured performance metric.

Chapter 5: Experimental Platforms - The hardware and software systems used in the implementation of the above approach are explained. An outdoor navigational robot platform and a simulated robot platform (based on a physical, fielded robot) are described.

Chapter 6: Acting on Detected Inter-Layer Conflict - A basic implementation of the approach on a robot hardware platform is described. The description relates the formal definition to implementation details. Qualitative experimental results are presented.

Chapter 7: Learning Applied to Inter-Layer Conflict - Full-featured implementations of the approach on a simulated robot platform are described. Again, formal definitions are related to implementation details. Quantitative and qualitative results are presented.

Chapter 8: Future Directions - Remaining open questions and possible avenues of future research are explored.

Chapter 9: Conclusion - Concluding remarks are given.

Appendix A: Algorithms - Pseudo code implementations of algorithms developed in this dissertation are provided.

Appendix B: Experimental Data - Experimental data from experiments performed in this dissertation is provided.

CHAPTER II

BACKGROUND AND RELATED WORK

This work's primary contributions pertain to learning internal models of a robotic system's capabilities within hybrid robot control architectures, according to external system-level evaluation of the robot's performance. A large body of work exists dealing with each component of this topic, though the topic as a whole remains relatively unexplored. In an effort to define a base of knowledge on which to build this work, a review of background and related work is presented.

2.1 Dynamic and Kinematic Models

Two approaches to define constraints within the context of mobile robot navigation are closely related. The dynamic window approach and the use of configuration spaces both aim to model the dynamic and kinematic constraints of the robot.

Configuration spaces are geometric encodings of the robot's kinematic capabilities and constraints [12], [32]. The configuration space represents all achievable poses, defined at a certain level of abstraction, with respect to the robot's kinematic constraints, and, possibly, with respect to external constraints such as navigation obstacles. Changing the footprint or state space of the robot necessarily changes the configuration space of the robot, as it defines a different set of possible configurations. Because they are compact representations that are easily applied over large spatial scales, configuration spaces have been widely used in spatial reasoners and planners. However, while configuration spaces are a powerful representations of a robot's kinematic constraints and are applicable over large spatial scales, they do not offer any information with respect to a robot's dynamic constraints.

The dynamic window approach is a technique that accounts for the robot's dynamic constraints within the reactive layer [17], [18]. It produces a sort of dynamic configuration space based on the robot's current translational and rotational velocity, and the robot's maximum translational and rotational accelerations. Because of its reactive nature, it works on over a short temporal and spatial scale. Over this small scale, the dynamic window approach evaluates the effects of a set of discretized control outputs with respect to its local spatial environment. In [10], the dynamic window approach is used in conjunction with a local objective function and partial global path planning. While this approach offers a powerful model of the robot's dynamic constraints, it is only applicable only over a small temporal and spatial scale.

2.2 Behavior-Based Robotics

Over the past two decades, the field of behavior-based robotics has developed an impressive body of research. The behavior-based approach to robot control is predicated on motor control that is tightly coupled to perception, without significant reasoning over world models. Brooks [11] and Arkin [3] developed early implementations of behavior-based architectures.

Behavior-based systems differ from AI-based "sense-plan-act" systems and control theoretic systems in several important ways [4]:

Reactivity - The tightly coupled perception to motor control loop eschews significant reasoning over complex world models, allowing the robot to respond quickly to changes in the environment and dynamic components of the environment.

Isolation - Individual behaviors work only to serve a particular interest within the robot's total scope of applicability. That is, a behavior that is designed to keep the robot clear of static obstacles in the environment does not attempt to also guide the robot toward any other navigation goals, such as staying on a paved road.

Composability - Behaviors are usually used in combination. An arbitration process is used to either select a motor command from those provided by the behaviors [11], select a motor command that is a combination of those provided by the behaviors [3], or select a motor command that is mutually agreeable upon by the behaviors [41].

Emergence - The system-level performance of the robot is product of the interaction of the behaviors over time, rather than the product of one prescriptive function.

Robustness - Because behavior-based systems make few assumptions about the world, it is less likely that these assumptions will be violated.

Much of this work makes use of a behavior-based architecture based on the work of Rosenblatt [41], [42]. The Distributed Architecture for Mobile Navigation (DAMN) was developed partly in conjunction with the NavLab project. It implements behavior-based control in a voting-based scheme. Individual behaviors express their interests by distributing an allocation of "votes" over candidate motor outputs. Each behavior's allocation of votes can be adjusted to effect a gain weighting scheme. An arbitrator sums the votes cast for each candidate motor output, and selects the candidate with the highest vote total.

2.3 Hierarchical and Hybrid Control Architectures

While certainly powerful, purely reactive behavior-based architectures have drawn criticism for their concentration on the local frame. This has led many to argue for the integration of reactive behavior-based systems with control layers that work toward higher-level goals. In this paradigm, higher-level goal oriented deliberative layers provide input to lower-level reactive layers. This input may take many forms, such as commanded local goals (e.g. a local waypoint), or prescribed modes of operation (e.g. a set of behaviors to activate or tasks to accomplish).

Arkin's AuRA architecture [5] and Gat's Atlantis architecture [21] are early examples of hybrid deliberative-reactive architectures. In both, the reasoning done by the deliberative layer is fundamentally different from that done by the reactive layer. The deliberative layer works to achieve global goals based on world models. The reactive layer works to achieve local constraints based on current sensor input. Each architecture suggests methods for combining the globally-based deliberative input with the locally-based reactive reasoning.

In [22], Gat argues that three-layered architectures consisting of a deliberative layer, a sequential layer, and a reactive layer provide a natural division of labor, loosely corresponding to layers dealing with the robot's future, past, and present, respectively. The deliberative layer plans paths or actions, reasoning over the robot's future. The sequential layer maintains the robot's state within a sequence of tasks to be done, corresponding to a representation of the what the robot has already done, i.e. the robot's past. Finally, the reactive layer reasons over the robot's current sensor input; the robot's present state.

Layered architectures have been especially successful in the area of field robotics. Albus' 4D-RCS architecture [1] builds upon the concept of layered architectures, implementing an extended hierarchy of layers. 4D-RCS was designed explicitly for distributed multi-agent military operations and provides for hierarchical layers corresponding to traditional military organization (e.g. battalion layer, platoon layer).

Recently successful field robotics platforms have also made use of hierarchical and hybrid control systems. Stanley, the robot winner of the DARPA Grand Challenge [49] made use of a global trajectory planner advising a lower-level real-time controller to navigate 120 miles of desert. The winner of the DARPA Urban Challenge, Boss [52], made use of an architecture that included a global task planner, a local path planner and a local behavior-based design to navigate over 60 miles of urban terrain.

2.4 Consistency within Hierarchical Control Architectures

Because the layers of hierarchical and hybrid control systems exploit fundamentally different reasoning biases, care must be taken to ensure that different layers are, in fact, working toward the same system-level goals. That is, designing a system with a deliberative layer that delivers global trajectories to a reactive layer that is unable to act upon the trajectories does not achieve system level goals of smoothly reaching a spatial goal in a short amount of time. Often, implementers rely on empirical results to "fine tune" system-level performance.

However, some attempts have been made to guarantee certain performance metrics under specific assumptions. In [37], Pappas presents a mechanism to guarantee consistency with a hierarchical control system. The presented strategy is to generate a continuous mapping of the state spaces used by each layer in the hierarchy. Doing this, it is possible to prove (or disprove) that any trajectory produced by a higher layer is achievable by lower levels. However, this technique is only applicable to hierarchical continuous linear control systems, and does not apply to hybrid or behavior-based control systems.

2.5 Learning within Hierarchical Control Architectures

Because of the difficulties associated with assuring system-level performance within hierarchical control systems, a body of work has evolved promoting learning across layer boundaries or across task decompositions.

Early work in reinforcement learning across task decomposition was done by Lin [33]. In Lin's work, the system designer decomposed the robot's task into low-level skills and high-level skills (which make explicit use of the low-level skills) that the robot will need to complete the task. Q-learning [46] was used to first learn the low-level skills, then the high-level skills.

Similar to Lin's work, Stone [43] implemented an approach to task decomposition

and learning within the context of robot soccer. Rather than using reinforcement learning at all layers, Stone relied on human insight to choose appropriate learning techniques at each layer. Higher-level layers were learned making explicit use of learned low-level layers. Stone extended this hierarchy up to the team level and demonstrated its effectiveness in several international competitions.

In [6], Balch demonstrated the use of reinforcement learning for robots to learn a sequential layer strategy in the form of a finite state automata (FSA), based on a designer-implemented reactive layer. Balch implemented a set of behavioral assemblages, and defined states in an FSA mapping to each behavioral assemblage. He then used Q-learning to learn transitions between the states in the FSA.

2.6 Evaluation of Robotic Navigation Systems

The evaluation of robotic navigation systems has been driven by funding agencies and industrial developers to include a wider scope of metrics than simply time-to-goal or distance-travelled. While this field is still developing we can look to the management of major programs for direction in our evaluation of systems.

The DARPA Learning Applied to Ground Robots (LAGR) program evaluated experimental robotic systems in outdoor, unstructured environments [13]. In its evaluation of participants, DARPA emphasized three metrics:

- achievement of goal
- time to goal vs. a baseline system
- improvement in time to goal after learning about the environment

Another DARPA program, the DARPA Urban Challenge evaluated experimental robotic systems in an outdoor, urban environment [14]. The Urban Challenge was set up as a race, with all competitors on the course at once. In evaluating the competitors, DARPA emphasized four metrics:

- achievement of goals
- time to goals
- adherence to human traffic laws
- safety with respect to other traffic and property

The LAGR evaluation sets a precedence for evaluation of learning in field robotics. That is that the robot should not only improve its time-to-goal performance with learning, but it should also perform better than a baseline system. The Urban Challenge evaluation demonstrates a need for qualitative evaluation in navigation tasks. Indeed, a robot could have completed the race by driving through lawns and cutting off other traffic, but that would have been an undesirable result. When evaluating robotic navigation systems, it is important to remember that it is not only important how fast the robot gets to the goal, but also *how* it goes about getting there.

2.7 Discussion

This section has touched on several important bodies of research with respect to this work. While it has certainly not covered them exhaustively, several important points were covered:

- Powerful models of robotic kinematic and dynamic constraints exist, but are not universally applicable.
- Behavior-based robotics makes use of primitive behaviors, combined by an arbitrator, to implement fast, flexible robot navigation.
- Hierarchical and hybrid architectures divide the tasks of deliberative and reactive control into separate modules with fundamentally different biases toward the task of robot navigation.

- The problem of rectifying the biases of the deliberative and reactive layers is implicitly performed by all hierarchical and hybrid control architectures, but is only explicitly possible under specific assumptions.

- Hierarchical and layered learning has been implemented successfully on complex robotic systems.

- Explicitly learning across hierarchical layers has been successfully demonstrated between sequential and reactive layers.

- The evaluation of robotic navigation systems cannot be limited to simple time-based metrics, but must also take into account improvement by learning and human rules of conduct.

This work builds upon the concepts of hierarchical and layered learning to rectify kinematic and dynamic models used by different layers of hybrid hierarchical control systems. This is different from work by Balch or Stone in that it explicitly works to detect differences in the respective layers' models of the robot, act with respect to these detected differences, and adjust the models with respect to external evaluation of system-level performance. While the presented implementations obviously make specific choices within the deliberative and reactive layers, care is taken to present the approach in a general manner, applicable to an array of deliberative and reactive techniques.

CHAPTER III

HYBRID ARCHITECTURES AND INTER-LAYER CONFLICT

3.1 Introduction

As discussed in Chapter 1, this work addresses one main and two subsidiary research questions:

Research Question - In the context of spatial navigation, and the scope of the interface of the deliberative and reactive layers, how can conflict between the models of the capabilities of a robot used in the layers of a hybrid deliberative-reactive robot control architecture be detected and corrected to improve overall system-level performance?

Sub-Question 1 - How can conflicts between the deliberative and reactive layers' kinematic and dynamic models of the robot's relationship to the world be detected?

Sub-Question 2 - How can the robot learn from these detected conflicts to change its model of its relationship to to the world, in particular within the deliberative layer?

In this chapter, we elaborate on the research question and discuss a high-level approach to the question. Briefly, the approach discussed is to first define a measurement signal representing the measured system-level performance of the robot as it functions in in the world. Then, a learning mechanism is introduced, learning a concept relating the plan, as created by the deliberative layer, to an expected system-level performance, as executed by the reactive layer, in the context of the robot's

environment. This learned concept is then used by the deliberative layer to adjust not its model of the world, but its process for generating plans for the robot within the world.

This approach is explained in more depth in the following sections. First, we discuss instances of and reasons for conflict between layers of a hybrid control architecture, in the context of spatial navigation applications. Then, we discuss the detection of conflict in these architectures. Finally, we set up a high-level approach for correcting these conflicts. This leads into Chapter 4, which sets up a precise, formal representation for the problem and approach.

3.2 Inter-Layer Conflict

Hybrid control architectures often use different levels of abstraction at different levels of operation. For example, layers lower in the hierarchy, such as a behavior-based reactive layer, operate at a high spatial resolution, high temporal resolution, small spatial scope, and short temporal scope. Layers higher in the hierarchy, such as a global path planner-based deliberative layer, operate at a comparatively low spatial resolution, low temporal resolution, large spatial scope, and long temporal scope. Additionally, to deal with the computational complexity of reasoning over large spatial and temporal scopes, higher-level layers often work in a reduced state space. This leaves the lower-level layers to "fill in the blanks" with respect to these unused dimensions as the robot's execution progresses. The differences in abstraction and state space between these encapsulated layers can result in conflicting models of the robot's relationship with the world. To develop an intuition for the occurrence of conflicting models, hypothetical examples of conflict in kinematic and dynamical models between deliberative and reactive layers are presented. While these examples are representative of the types of conflicts addressed in this work, they, by no means, span the space of all possible conflicts.

3.2.1 A Kinematic Example

As a hypothetical example of a conflict in kinematic models, imagine a robot shaped like a long bar, with a center of rotation at its center of mass (Figure 1(a)). This robot moves about in a planar world, but has three degrees of freedom, x, y, and θ. If designing a two-layer deliberative-reactive control architecture for this robot, a designer might start by implementing a behavior-based reactive controller, capable of reasoning over all three degrees of freedom. Then, in an effort to make the planning problem more tractable, the designer might implement a two-dimensional grid-based A* path planner [32].

Projecting the robot's three degree of freedom configuration space into the two degree of freedom planner poses a dilemma for the designer. The designer could choose a "pessimistic" view of the robot's configuration space, ensuring the robot can turn in place anywhere suggested by the planner, as shown in Figure 1(b). Alternatively, the designer could choose an "optimistic" view of the robot's configuration space, only ensuring that the robot can move in the translational direction anywhere suggested by the planner, as shown in Figure 1(c). Because the pessimistic configuration space will likely exclude the robot from many possible navigation paths, there is ample reason to choose the optimistic configuration space for the planner, and leave the job of rotating the robot appropriately to the reactive controller.

This design would likely work in a many environments, but consider the environment shown in Figure 2. The low-dimensional configuration space used by the global planner perceives the narrow corner as navigable by the robot. However, the high-dimensional configuration space used by the reactive controller perceives this the corner as unnavigable, since the robot does not have enough room to turn in the narrow corridor. Conflict and deadlock have occurred. While some architectures have proposed mechanisms to re-plan in the event of deadlock (e.g. cognizant failure [23]), using an updated world model, in this case, the world model is still correct.

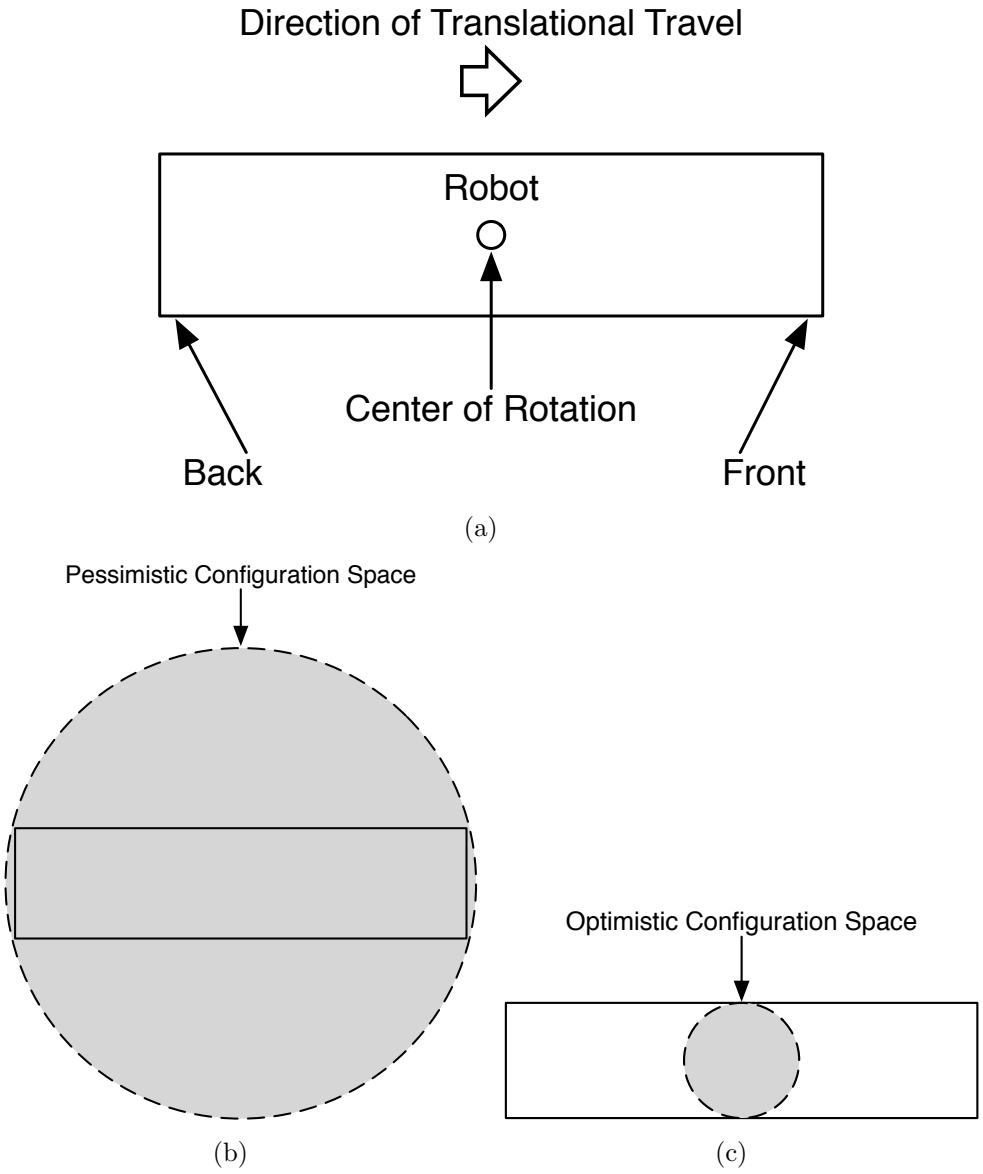


Figure 1: A hypothetical, bar-shaped robot is shown in (a). Its "pessimistic" two-dimensional configuration space is shown in (b). Its "optimistic" two-dimensional configuration space is shown in (c).

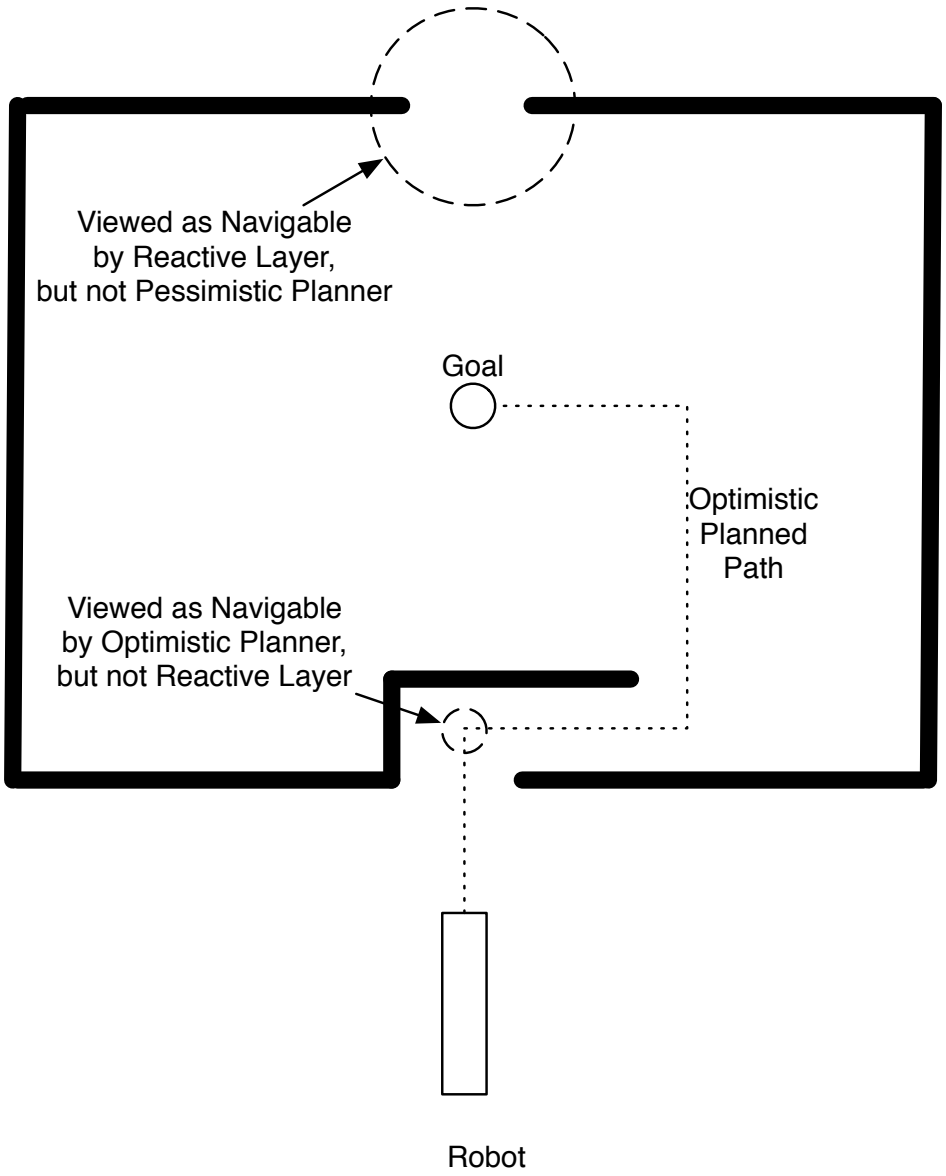


Figure 2: A depiction of an environment in which the bar-shaped robot's optimistic configuration space used by the planner conflicts with the three-dimensional configuration space used by the reactive controller. The shortest path found using the optimistic configuration space is not actually navigable by the robot. Note, however, that no path would be found using the pessimistic configuration space.

Re-planning will result in the same plan. What is at conflict is the deliberative and reactive layers' kinematic models of the robot. To obtain an achievable plan the planner must re-plan based on a different model of the robot's relationship with the world.

3.2.2 A Dynamical Example

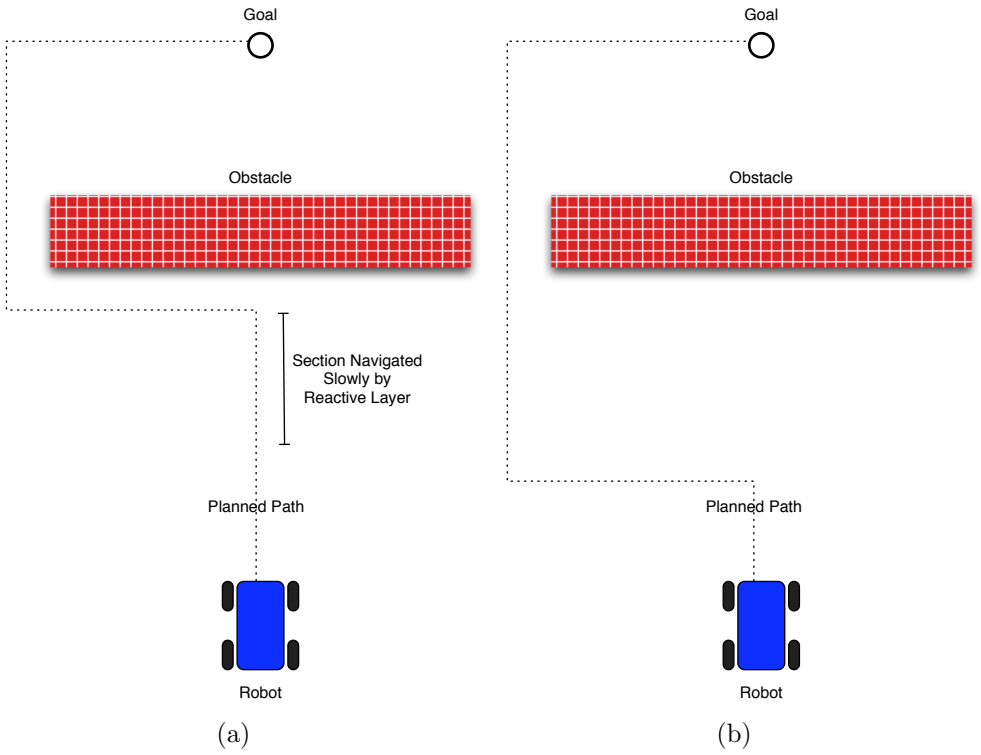


Figure 3: (a) depicts a fastest-path plan based on a baseline dynamical model of the robot with respect to the world. The robot will move slowly as it approaches the wall, because the the reactive layer slows the robot down as it approaches navigation obstacles. (b) depicts a fastest-path plan based on a more-informed dynamical model of the robot's relationship with the world. The robot does not need to slow down with respect to the wall because it never approaches the wall.

In another hypothetical example, consider a car-like robot, capable of high-speed navigation in relatively open terrain (Figure 3). Again, it makes use of a two-layer deliberative-reactive control architecture. Its reactive layer outputs commanded curvature (i.e. steering angle) and velocity set-points. It uses an approach in the style

of the dynamic window approach [17] to choose curvatures and velocities that are allowable given its current sensor input. Its deliberative layer is implemented as a two-dimensional grid-based A* path planner. For ease of illustration, assume that the planner works over a four-connected grid.

Given that the robot's task is to move from its starting location to the goal location as quickly as possible, consider the proposed plans in Figures 3(a) and 3(b). In Figure 3(a), the plan takes the robot straight at the wall before turning left and navigating around the wall. As this plan is executed, the robot's reactive controllers will slow the robot down as it approaches the wall. While this plan does, in fact, get the robot to the goal location, it does not get it to the goal location as quickly as the plan presented in Figure 3(b). This plan is of equivalent length, and turns the robot around the wall well before the reactive controllers would slow the robot down.

In this example, the deliberative layer that created the plan if Figure 3(a) has encountered a conflict between its dynamical model and the manner in which the reactive layer executes its plans. It generated a plan for getting the robot to the goal in the minimum time. However, when executed by the reactive layer, the plan was significantly slower than other possible plans. It is the aim of this work to develop mechanisms to detect, act with respect to, and learn with respect to this type of inter-layer conflict.

3.3 Detection and Correction of Inter-Layer Conflict

In order for the robot to improve its performance with respect to inter-layer conflicts, it must first detect these conflicts. To detect conflict between two architectural layers, we propose the following strategy: First define the robot's execution in terms of unique, independent experiences. Then, compare the high-level layer's model of the outcome of each experience to the measured system-level performance within that experience. Finally, modify the high-level layer with respect to the difference in its

expectation of outcome and the measured outcome.

As mentioned earlier, generally, the reactive layer of a hybrid control architecture works at a high spatial resolution, high temporal resolution, small spatial scope, and short temporal scope, while the deliberative layer works at a comparatively low spatial resolution, low temporal resolution, large spatial scope, and long temporal scope. Due to this tendency, the reactive layer is more likely to be derivable from first principles, or at least less likely to rely on coarse heuristics than the deliberative layer. Because of this divide, we propose to modify the deliberative layer when conflict is detected, rather than the reactive layer. The deliberative layer's model of the robot's functionality is more removed from its physical grounding, and holds more room for error (and therefore more room for improvement).

We will define an experience as the unit of comparison of an architectural layer's model of execution and the externally measured system-level performance of the robot. Experiences should be defined within the scope of the layer which will be modified. Experiences should also divide the robot's execution into independent non-overlapping segments. Basing the definition heavily on accepted models of supervised learning, the experience is made up of four parts:

command - the high-level layer's output to lower layers

end criteria - a logical criteria for ending the current experience and starting another

context - a representation of the state of the system that was used for creating the command

measured outcome - the measured value of an external metric relating measuring the success of the command at a system level

These parts will have different instantiations in the scope of different functional

modules. For example, a waypoint-based navigation planner might define an experience as:

command - the next waypoint

end criteria - achieving the waypoint, or moving 10 meters away from its position at the beginning of the experience without achieving the waypoint (i.e. failure to achieve the waypoint)

context - the local map around the next waypoint (a section of occupancy grid)

measured outcome - success in achieving the next waypoint (a binary function)

After each experience has been completed (or, if batch processing is necessary, after many experiences have been completed) the architectural layer's model of the measured outcome is modified based on the completed experiences. Staying with the waypoint-based navigational planner example, the expected outcome would be to achieve the next waypoint. If, after moving 10 meters away from its position at the beginning of the experience, it has not achieved the next waypoint, the measured outcome would be that it failed to achieve the next waypoint. The correction signal to the planner would consist of the next waypoint (the command), the local map (the context), and the measured outcome (one bit, marking the experience as failed).

Once the correction signal is defined, it is up to the respective functional modules to interpret the correction signal and modify their models of the system-level effect of their output on the robot. In the example of the waypoint-based planner, the planner might take the failed experience and label that particular location in its map as intraversable. This would meet the minimum requirement of modifying its model of the the robot's relationship with the world. Alternatively (and more generally), it might use that failed experience as input to a learning algorithm to learn to label similar situations as intraversable.

3.4 Learning from and Acting on Inter-Layer Conflict

While detecting, remembering and acting on instances of hierarchical conflict can be used to improve a robot's system-level performance, the improvement does not carry over to new situations. If it is the designer's goal to design a system that performs well in novel environments, using learning algorithms to generalize about detected hierarchical conflicts would be a good option. A natural way to generalize over this signal is to formulate it as a supervised learning problem.

Supervised learning can be viewed as learning to predict the value of a function for a set of arguments, given a number of examples of arguments and the corresponding output of the function [35]. Two categories of supervised learning algorithms are of interest in this context: regression algorithms (for real-valued functions, such as speed) and classification algorithms (for finite set-valued functions, such as success vs. failure). Regression algorithms include linear regression, artificial neural networks, and k-nearest neighbor regression. Classification algorithms include decision trees and support vector machines.

In the context of a supervised learning approach, high-level layers view low-level layers as functions that take as arguments a command and context (as defined in the experience, above) and return a value in the form of an externally measured outcome. If the high-level layer can better predict the value of this outcome (the expected outcome of the experience), by generalizing over completed experiences and their measured outcome, it will encounter less conflict and produce better system-level performance.

Building a simple model of the functionality of a hybrid control architecture, define p as the output of the deliberative layer of an architecture, i.e. a plan. Also, define $c(p, m)$ as a model of the cost or performance of the proposed plan, p, within a context, m, i.e. a map, and $h_c(p, m)$ as a measurement function, measuring the cost or performance as p is executed. In the context of the above defined learning

experience, p corresponds to the command, m to the context, and $h_c(p, m)$ to the measured outcome.

Within the context of spatial navigation, most deliberative layers work to provide a plan that minimizes the associated cost, given the current map. That is, the chosen plan at time t, p_t, is the plan within the space of all plans, P, that minimizes the model of the associated cost, given the map at time t, m_t,

$$p_t = \arg\min_{p \in P} \; c(p, m_t). \tag{1}$$

Clearly, the "optimal" plan would be one that minimizes not the cost *model*, but the actual measured cost,

$$p_t^* = \arg\min_{p \in P} \; h_c(p, m_t). \tag{2}$$

If we were to improve the cost model such that it approached the the measured cost, optimizing over the cost model provides the "optimal" plan,

$$p_t^* = \lim_{|c(p,m_t) - h_c(p,m_t)| \to 0} \arg\min_{p \in P} c(p, m_t). \tag{3}$$

Since it is not possible to know the measured cost a priori, this sets up a clear learning task: to choose a plan to minimize the cost model, given that the cost model is based on previous measured experiences,

$$p_t = \arg\min_{p \in P} \; E[c(p, m_t)] \mid \{h_c(p_t, m_t)\}^t. \tag{4}$$

This learning problem will be expanded in more detail in Chapter 4.

3.5 Discussion

Conflict between layers of a hybrid control architecture can prevent robotic navigation applications from achieving their commanded goals. This chapter provides hypothetical examples of inter-layer conflict and demonstrates the loss of performance caused by the conflict. It also proposes a high-level approach to detecting and learning with respect to conflict between layers. It is proposed that correcting or accounting for

inter-layer conflict is a feasible way to improve system-level performance. It is also proposed that using supervised learning algorithms to generalize about experiences provides additional leverage to improve the robot's performance. Certainly, at this point, the proposed approach is still lacking in detail and obviously subject to substantial implementation considerations. In the next chapter, this approach is formalized in the language of hybrid control theory, and described further. Later chapters will demonstrate the approach in action and detail the implementation of the approach.

CHAPTER IV

REPRESENTATION

4.1 Introduction

As expressed in Chapter 1, the goal of this work is to explore ways in which conflicts between the deliberative and reactive layers of hybrid robot control architecture can be detected and used to improve the robot's system-level performance. One of the defining characteristics of this type of architecture is the separation of the discrete, forward-looking planner, in the deliberative layer, from the continuous, reactive controller, in the reactive layer. While this separation of functionality makes understanding each layer easier, understanding and modeling the interaction between the layers remains a problem. In approaching this problem, we will first develop a formal representation of the architecture.

In choosing a formalization in which to express this problem, the field of hybrid control theory is a natural fit. Hybrid control theory's canonical expression, the hybrid automaton, provides a mathematically sound and intuitive formalization for expressing the interplay between discrete and continuous dynamics. In this chapter, we review the notation used in reference to the hybrid automaton, propose an expression of a deliberative-reactive architecture within this notation, and define the task of learning models of continuous execution of discrete plans to improve system-level performance.

4.2 Hybrid Control Theory

4.2.1 Background

Hybrid control theory has been developed over the last two decades to provide a mathematically sound model for studying systems that are influenced by both continuous and discrete dynamics. The model of the hybrid automaton has been used to model such diverse phenomena as transportation, biological and robotic systems [27] [28] [8]. Aside from the shear descriptive power, hybrid control theory also allows for both qualitative and quantitative analysis of these systems. Under certain conditions, these systems can be analyzed for safety, completeness, reachability and optimality [25] [20] [16] [15].

Hybrid control theory also offers advantages over other representations, such as discrete event systems. First, hybrid control theory easily handles a wide range of controllers, including controllers without any explicit intentionality. Whereas a discrete event system creates plans that are driven by the success or failure of specific sub-goals, hybrid control theory allows for the use of generic controllers and multiple paths of execution. While formal reachability analysis can prove that an instantiation is capable of reaching a desired state, the execution path is not defined a priori. Additionally, the continuous controllers used in hybrid control are grounded in physical systems. This provides a clear path for the definition of measurement and supervisory signals. This combination of expressiveness and analytical power makes the hybrid automaton an ideal model for expressing the functionality of the deliberative-reactive control architecture.

4.2.2 Representation and Execution

Over the years, a number of representations have been proposed for hybrid control systems [9], [50], [34]. Erring on the side of clarity and applicability, we will use a representation of hybrid systems that closely follows the representation developed in

[51].

In this work, we will define a controlled hybrid automaton H as a collection

$$H = (Q, X, Init, \Sigma, \Upsilon, f, Dom, Reset, g),$$

where

- $Q \cup X$ is a collection of state variables. Q is a finite set of discrete states. X is a finite set of real-valued states. In this work, we will let $X = \mathbb{R}^n$
- $Init \subseteq Q \times X$ is a set of initial states.
- $\Sigma = \Sigma_U \cup \Sigma_D$ is a finite set of discrete inputs. Σ_U is a finite set of discrete control inputs. Σ_D is a finite set of discrete disturbance inputs.
- $\Upsilon = U \cup D$ is a finite set of continuous inputs. U is a finite set of continuous control inputs. D is a finite set of continuous disturbance inputs.
- $f : Q \times X \times \Upsilon \to TX$ is a controlled dynamical system which describes the evolution of x in each discrete state $q \in Q$, given the continuous control and disturbance inputs $\upsilon = (u, d) \in \Upsilon$.
- $Dom \subseteq Q \times X \times \Sigma \times \Upsilon$ is a domain, defining the conditions under which continuous evolution may continue, within each discrete state. When the system exits Dom, the state of the system is updated according to $Reset$.
- $Reset : Q \times X \times \Sigma \times \Upsilon \to Q \times X$ is a reset function which encodes the discrete transitions of the hybrid automaton. $Reset$ defines the state of the system when it exits Dom, updating both the discrete and continuous states.
- $g : Q \times X \to \Sigma_U \times U$ is a controller for the controlled dynamical system, f and the reset function $Reset$. The controller takes in the state of the automaton, and feeds back control inputs for both the discrete and continuous domains. In

this way, the controller may affect not only the continuous progression of the dynamical system through the input U, but also the progression of the discrete reset function through the input Σ_U.

We may refer to the tuple $(q, x) \in Q \times X$ as the *state* of the hybrid automaton. Likewise, we may refer to the tuple $(\sigma_U, u) \in \Sigma_U \times U$ as the *control input* of the hybrid automaton and $(\sigma_D, d) \in \Sigma_D \times D$ as the *disturbance input* of the hybrid automaton.

Executing within this representation, the system begins in an initial state, $(q_0, x_0) \in \mathit{Init}$. While the state remains within the domain of q_0, the dynamical system progresses continuously through the state space. $q(t)$ remains constant and $(\sigma_U(t), u(t))$ is the output of the controller, $g(q_0, x(t))$. $x(t)$ evolves over time as the solution to the differential equation

$$\dot{x} = f(q, x, u, d).$$

This continuous evolution continues until the state (possibly) exits the domain, i.e. $(q, x, \sigma, \upsilon) \notin \mathit{Dom}$.

When the state exits the domain, the system evolves discretely. The time interval I between discrete transitions is denoted as

$$I_i = [\tau_i, \tau_i'],$$

where τ_i marks the beginning of the interval (i.e. the time that the system made its ith discrete transition) and τ_i' marks the end of the interval (i.e. the time that the system made its $i+1$th discrete transition). Because the automaton's discrete state transitions instantaneously, τ_i' always equals τ_{i+1}. (i.e., the time intervals are closed and overlapping at the boundaries.) The discrete state may transition multiple times in an instant, or persist for some positive amount of time. Therefore, the following relation holds:

$$\tau_i \leq \tau_i' = \tau_{i+1}.$$

Coinciding with the discrete transition, the state progresses instantaneously according to the reset function,

$$(q(\tau_{i+1}), x(\tau_{i+1})) = Reset(q(\tau_i'), x(\tau_i'), \sigma(\tau_i'), \upsilon(\tau_i')).$$

The reset function progresses the discrete state from $q(\tau_i')$ to $q(\tau_{i+1})$ and the continuous state from $x(\tau_i')$ to $x(\tau_{i+1})$. If, according to the reset function, the state $(q(\tau_i'), x(\tau_i'))$ may map to more than one new state, the discrete input, $\sigma(\tau_i')$, may be seen as "choosing" between multiple possible reset relations. While the representation allows the continuous state x to be reset arbitrarily, when the continuous state is based on a physical system that cannot instantaneously jump between states, the reset relation will be usually be constrained such that

$$x(\tau_{i+1}) = x(\tau_i').$$

The standard interpretation of hybrid automata is that they *accept* executions, rather than *generate* executions. Therefore, along with the definition of hybrid automaton, H, we define a *trajectory acceptance condition*, Acc. We denote the sequence $\tau = \{I\}_{i=0}^{N}$ as the *hybrid time trajectory*, and the collection $\chi = (\tau, q, x, \sigma, \upsilon)$ as the *hybrid execution*. The trajectory acceptance condition is a map from the set of all executions to $\{True, False\}$

$$Acc : \chi \rightarrow \{True, False\}$$

This condition may be interpreted in a number of ways. Commonly, in the study of hybrid automata, achieving the acceptance of a hybrid execution may be interpreted as achieving some performance metric, such as safety (i.e. the system remains within the domain throughout its execution). From a more applied standpoint, achieving acceptance may be interpreted as the successful completion of a task. Depending on its implementation, the automaton may stop execution after acceptance, or continue indefinitely.

4.2.3 Graphical Representation

While the mathematical representation of hybrid automata provides an excellent tool, with its conciseness and completeness, it is not always intuitive for system designers to think in these terms. For this reason, it is often convenient to think of hybrid automata in terms of a graphical model. In fact, this representation has become so common, that it has become the canonical representation in many circles [2] . Here, we present the above representation of a hybrid automata as a graphical model. This model will later be used to relate the general model of a hybrid automata to the specific implementation of a hybrid deliberative-reactive control architecture.

An automaton can be represented as an augmented directed graph,

$$\Gamma = (V, E, \mathit{Init}, F, G, \mathit{Dom}, \mathit{Guard}, \mathit{Reset}).$$

The vertices of the graph, V, correspond directly to discrete states. The edges, $E \subseteq V \times V$ correspond to possible transitions between the discrete states. Using this graph representation, shown in Figure 4, as a skeleton, we can fill in the components of the hybrid automaton.

For each discrete state, $q \in Q$, we assign the following:

- v_q, a unique vertex in V.
- Init_{v_q}, a set of initial continuous states for each discrete state,

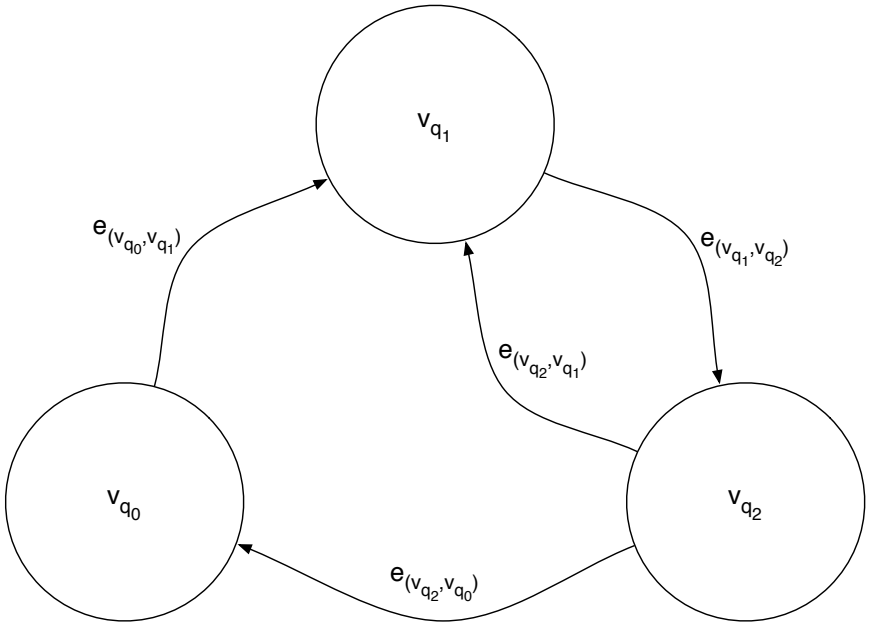
$$\mathit{Init}_{v_q} = \{x \in X \mid (q, x) \in \mathit{Init}\} \subseteq X,$$

  such that

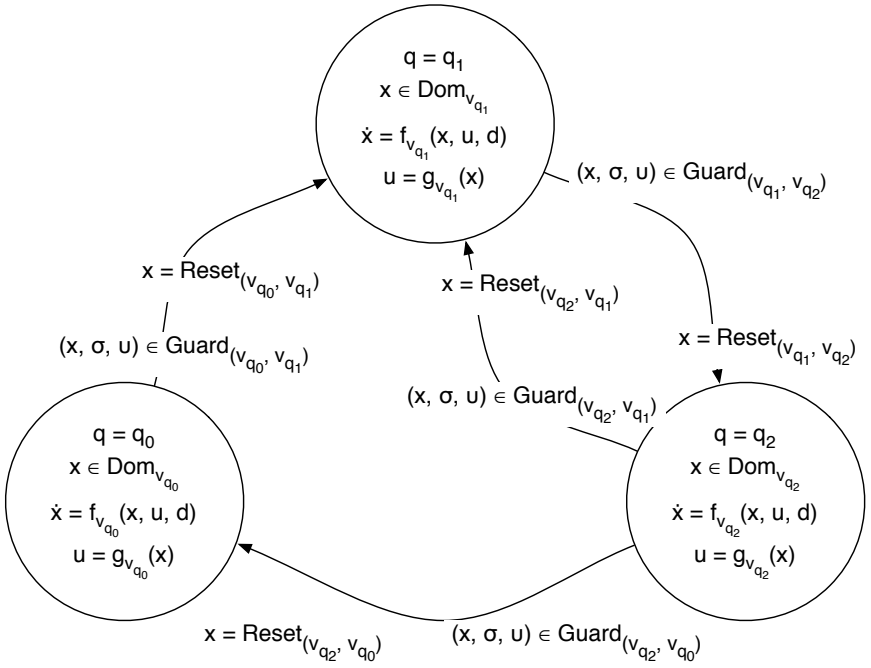
$$\mathit{Init} = \bigcup_{v_q \in V} \mathit{Init}_{v_q}.$$

- f_{v_q} a differential equation,

$$f_{v_q}(x, \upsilon) = f(q, x, \upsilon),\ F = \{f_{v_q}\}_{v_q \in V}$$



(a)



(b)

Figure 4: In (a), a simple graph structure showing three states (the vertices) and the possible transitions between them (the edges). In (b), the same graph structure is augmented with the mathematical model of a controlled hybrid automata.

- g_{v_q}, a controller,

$$g_{v_q}(x) = g(q, x),\ G = \{g_{v_q}\}_{v_q \in V}$$

- Dom_{v_q}, a domain,

$$Dom_{v_q} = \{(x, \sigma, \upsilon) \in X \times \Sigma \times \Upsilon \mid (q, x, \sigma, \upsilon) \in Dom\} \subseteq X \times \Sigma \times \Upsilon,$$

such that,

$$Dom = \bigcup_{v_q \in V} Dom_{v_q}.$$

For each possible transition between the discrete states, we create a unique edge in E, such that,

$$E = \{(v_q, v_{q'}) \in V \times V \mid (q', x') \in Reset(q, x, \sigma, \upsilon)\} \subseteq V \times V,$$

for some $x,\ x' \in X,\ \sigma \in \Sigma$, and $\upsilon \in \Upsilon$. We can then associate with each edge, $e_{(v_q, v_{q'})} \in E$:

- $Guard_{(v_q, v_{q'})}$ a guard,

$$Guard_{(v_q, v_{q'})} = \{(x, \sigma, \upsilon) \in X \times \Sigma \times \Upsilon \mid (q', x') \in Reset(q, x, \sigma, \upsilon)\} \subseteq X \times \Sigma \times \Upsilon,$$

for some $x' \in X$.

- $Reset_{(v_q, v_{q'})}$ a reset relation,

$$Reset_{(v_q, v_{q'})}(x, \sigma, \upsilon) = \{x' \in X \mid (q', x') \in Reset(q, x, \sigma, \upsilon)\} \subseteq X.$$

The execution of the automata is similar to the above described execution. The automata begins in an initial state, $(v_{q_0}, x_0) \in Init_{v_{q_0}}$. It progresses continuously according to f_{q_0} and g_{q_0}, until the state leaves Dom_{v_q} at time τ_0'. At this point, it transitions to another state, $v' \in V$, along some edge, $e_{(v_{q_0}, v_{q'})} \in E$, such that, $(x(\tau_0'), \sigma(\tau_0'), \upsilon(\tau_0')) \in Guard_{(v_q, v_{q'})}$. Again, if more than one guard satisfies the transition out of $(v_{q_0}, x(\tau_0'))$, the discrete input $\sigma(\tau_0')$ may be seen as "choosing" between the possible transitions. The state is updated according to the reset function,

$x(\tau_1) = Reset_{(v_{q_0}, v_{q'})}(x(\tau_0'), \sigma(\tau_0'), \upsilon(\tau_0'))$ and $v_{q_1} = v_{q'}$. The evolution of the automata continues from the new state, $(x(\tau_1), v_{q_1})$.

The representation described in this section, while simple, is exceptionally powerful. As mentioned before, hybrid automata have been used to model and control a diverse set of systems in many domains. In the following section, we will model a hybrid deliberative-reactive robot control architecture as a hybrid automata. Additionally, we will identify functional components within this representation that may be improved, by detecting conflict and learning based on performance measurements, to improve system level performance.

4.3 Application to Hybrid Deliberative-Reactive Architectures

In Chapter 3, we described the problems associated with designing a hybrid deliberative-reactive robot control architecture, in the context of spatial navigation. Among those problems was an inherent disconnect between the deliberative and reactive layers that led to the generation of plans that were either sub-optimal or impossible, based on the capabilities of the reactive layer and the physical robot. In this section, we model this architecture as a hybrid automata, identify the functional components that are responsible for this disconnect and demonstrate how these components can be improved, using machine learning techniques over measured performance metrics.

Following the description of hybrid control architectures in Chapter 3, we divide the architecture into two distinct components, the reactive layer and the deliberative layer. The reactive layer is responsible for monitoring the robot's sensors, performing low-level navigation and decision making, and actuating the robot's motors. We model the robot's reactive layer as a continuous controlled dynamical system. The deliberative layer is responsible for integrating sensor input into maps, and planning routes and actions toward a given goal. We model the deliberative layer as a discrete process providing regular input to the reactive layer.

This model meshes well with the definition of the controlled hybrid automata in Section 4.2. In the following sections, we will model the architecture as a controlled hybrid automata. Using this representation, we will pose the spatial planning task as an optimization over the hybrid automata design space. Finally, we will demonstrate how learning the cost function used by the optimization from direct measurements on the reactive layer may result in lower-cost executions of the plan.

4.3.1 Reactive Layer

As previously mentioned, the reactive layer of the deliberative-reactive architecture handles the immediate sensory input and control output of the robot. We will use a subset of the model of the controlled hybrid automata to describe this component of the architecture. In particular, we will use the continuous components of the hybrid automata, f, g and Υ.

We begin by modeling the robot as a controlled dynamical system,

$$\dot{x} = f(q, x, \upsilon),\ q \in Q,\ x \in X,\ \upsilon \in \Upsilon$$

which exists in a world, W,

$$W \subset \mathbb{R}^d.$$

We are given a measurement function, h_y

$$h_y : X \to Y,\ Y = \mathbb{R}^n,$$

that provides access to the state of the robot, where $y \in Y$ is the measured state of the robot. Additionally, we are given another measurement function that gives sensory access to the state of the world, h_s,

$$h_s : X \times W \to S,\ S = \mathbb{R}^m,$$

where $s \in S$ is the collection of all observable sensory input:

$$s = \{h_s(x, w)\}_{w \in W},$$

for some $x \in X$. That is, s is a set of measurements made by the measurement function h_s from a position x of all observable points in the world, W.

Intuitively, s represents the robot's perception of the world via its sensors, and y represents the robot's perception of itself. Both s and y can be thought of as noisy or partially observable functions of w and x, respectively. For the purposes of representation, h_s and h_y can be simply thought of as pass-through functions, taking in one variable and passing on another. However, when we consider optimizing and learning over this representation, this distinction will become important.

Finally, we can close the loop in the reactive layer, defining the control input u as a function of the measurements of both the system state and the environment state. Thus,

$$u = g(y, s) \tag{5}$$

4.3.2 Deliberative Layer

In contrast to the reactive layer, the deliberative layer functions over discrete time and, rather than handle the immediate control of the robot, plans the control of the robot into the future to optimize some cost function. We will use the the discrete components of the hybrid automata to model the deliberative layer.

We model the robot's deliberative layer as a regularly updating discrete-timed event system which updates at times $t_0, t_1, \ldots, t_{final}$, where $t_i - t_{i-1} = \Delta t$, $\Delta t > 0$. These intervals account for the practical requirements of the execution of complex algorithms and management of large data sets.

Given that the robot is using a map to guide its path planning algorithms, we define the map M as an integrated set of sensory input. In each update cycle, the most recent set of sensory input, s_{t_i} is integrated into the map by the integration function m,

$$M_{t_i} = m(s_{t_i}, y_{t_i}, M_{t_{i-1}})$$

We assume m is a non-invertable function. That is, given M_{t_i} we cannot directly recover $\{(s_{t_0},\ y_{t_0}),\ (s_{t_1}\ , y_{t_1}),\ \ \ldots,\ (s_{t_i},\ y_{t_i})\}$. This is an important point, as given M_{t_i}, we cannot directly recover the reactive layer's output, $g(y_{t_i}, s_{t_i})$

Given an integrated representation of the world, the next step is to plan a control strategy through the world. Given that the world W is compact and connected, assume W is partitioned into a set of n regions, as shown in Figure 5,

$$R = \{r_i\}_{i=0}^{n}$$

such that

$$\bigcup_{i=0}^{n} r_i = W$$

and

$$r_i^o \bigcap r_j^o = \phi,\ \forall i, j,\ i \neq j$$

where $r^o \in R$ denotes an interior region. We designate one region as a goal region, r^o_{goal}, and another as a starting region, r^o_{start}.

For each region, we are given a collection of m control laws that may be employed in that region,

$$G_{r_i^o} = \{g_j(y, s)\}_{j=0}^{m},\ \forall r_i^o \in R$$

and a transition function,

$$d : R^o \times M \times G_{r^o} \rightarrow R^o$$

which provides a mapping from an interior region, a map, and a control law to the next region the control law will drive the robot toward. Intuitively, we can think of this is as the expected outcome of the control law. This mapping is important to the planning process as it provides a model of the outcome of the action of employing a particular control law. We are also given a cost model,

$$c : R^o \times M \times G_{r^o} \rightarrow \mathbb{R}^+$$

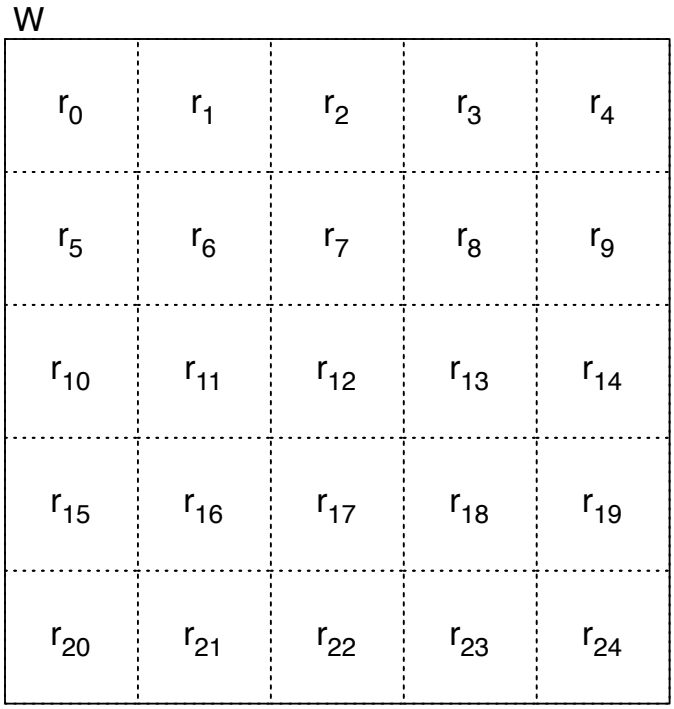


Figure 5: A diagram of the compact, connected world W, divided into regions r_0 through r_{24}. While W is shown in two dimensions and divided in a regular grid-like pattern, neither characteristic is a necessary condition of the presented representation.

that provides an expected cost of traversing region r, using the control law g, given the map M.

This representation is easily mapped into the same directed graph-based model used to represent the controlled hybrid automata, as shown in Figure 6,

$$\Gamma = (V, E, \mathit{Init}, F, G, \mathit{Dom}, \mathit{Guard}, \mathit{Reset}),$$

where:

- V is a set of vertices, directly corresponding to the set of interior regions,

  $$V = \{v_{r^o}\}_{r^o \in R^o}.$$

  The vertices, V, are defined in a one-to-one mapping with the regions, R^o, and represent the discrete states of the hybrid automata.

- E is a set of directed edges, $E \subseteq V \times V$. This set of edges corresponds to the connectivity described by the transition model,

  $$E = \{v_{r^o} \times d(r^o, M, g)\}_{r^o \in R^o,\ g \in G_{r^o}}.$$

- *Init* corresponds directly to the starting region, r^o_{start}. Since choosing a region in the world is equivalent to choosing a discrete state in the hybrid automata,

  $$\mathit{Init} = \{(v_{r^o_{init}}, r^o_{init})\}$$

- $F = \{f_{v_{r^o}}\}_{v_{r^o} \in V}$ is a set of differential equations describing the dynamics of the robot in each state. For many applications, the robot's dynamical behavior will be the same in all regions. However, as an example of an application that may contain multiple equations, one may consider an amphibious robot that behaves differently in regions that are made up of dry land and water.

- $G = \{g_{v_{r^o}}\}_{v_{r^o} \in V}$ is a set of controllers that provide input to the differential equation, $f_{v_{r^o}}$. G can be thought of as a set of controllers, one for each region, r^o, that have been selected for execution from G_{r^o},

$$G \subseteq \bigcup_{r^o \in R} G_{r^o}.$$

- *Dom* is a domain, defined such that,

$$Dom = \bigcup_{v_{r^o} \in V} Dom_{v_{r^o}}.$$

  While specific applications may put other constraints on the domain, if the robot is only planning over physical space,

$$Dom_{v_{r^o}} = \{(r^o, \cdot, \cdot)\} \subseteq X \times \Sigma \times \Upsilon.$$

- *Guard* is a set of guards that describe conditions for the possible transitions between states,

$$Guard_{(v_{r^o}, v'_{r^o})} = \{(r^{o\prime}, \cdot, \cdot) \in X \times \Sigma \times \Upsilon\}.$$

- *Reset* is a set of reset relations describing the instantaneous progression of the continuous state of the robot as it progresses from discrete state v_{r^o} to v'_{r^o}. Because x represents the physical state of the robot, it will not instantaneously evolve.

$$Reset_{(v_{r^o}, v'_{r^o})}(x, \sigma, \upsilon) = x.$$

As is common with many graph representations for planning, we augment the graph with a weight on each edge. l is a weight function,

$$l : E \to \mathbb{R}^+.$$

This function directly corresponds to the previously defined cost function, $c(r^o, M, g)$, where the edge corresponding to the weight is given by the transition model, $e =$

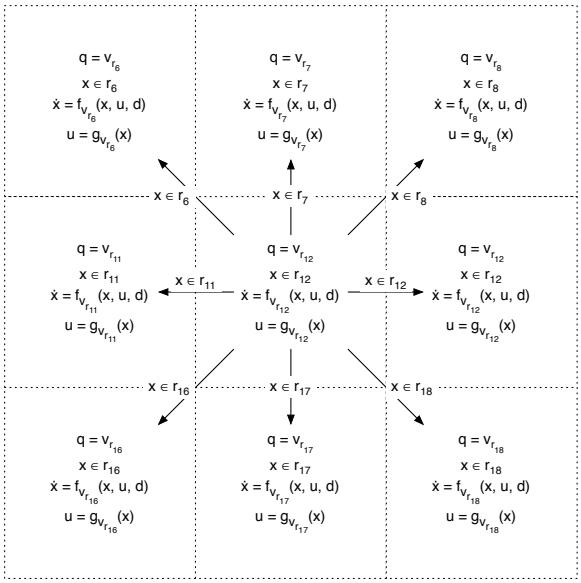


Figure 6: In (a), the center region (r_{12}) from the map in Figure 5 shown as a vertex in a graphical representation of a controlled hybrid automaton. A discrete state, domain, dynamical model and controller are associated with each region. A guard is associated with each possible out-going discrete transition into another region. (Incoming transitions are not shown.) In (b), the controller $g_{r_{12}}$ is depicted as a vector field in the region r_{12}. (This choice of controller implementation is for descriptive purposes and is not a necessary condition for the representation.) The transition model d predicts the next discrete transition to be made (in this case to r_{13}). The cost model c predicts the cost that will be incurred before the next discrete transition is made.

$(r^o, d(r^o, M, g))$,

$$l(e_{(r^o, d(r^o, M, g))}) = c(r^o, M, g).$$

Within this graph-based representation, the path planning problem can be defined as finding a path from $v_{r^o_{start}}$ to $v_{r^o_{goal}}$ along the edges in E to minimize the total cost incurred by following that path. The path from $v_{r^o_{start}}$ to $v_{r^o_{goal}}$ may be seen as a sequence of vertices,

$$\{v_{r^o_{start}}, v_{r^o_1}, \ldots, v_{r^o_{goal}}\}.$$

Since, the vertices are also states in the hybrid automata, the times of the transitions between these vertices makes up a hybrid time trajectory,

$$\tau = \{I_{r^o_{start}}, I_{r^o_1}, \ldots, I_{r^o_{goal}}\}.$$

The constraint that the trajectory must start in r^o_{start} and end in r^o_{goal} can be expressed in the trajectory acceptance condition,

$$Acc(\chi) = \begin{cases} True & \text{if } p(x(\tau_0)) \in r^o_{start},\ p(x(\tau_N)) \in r^o_{goal} \\ False & \text{else} \end{cases} \tag{6}$$

where $p(x) \in W$ is the position of the robot in the world.

To get from one vertex, $v_{r^o_i}$ to the next, $v_{r^o_{i+1}}$ the robot must execute a control law, $g \in G_{v_{r^o_i}}$, such that $d(r^o_i, M, g) = r^o_{i+1}$. Therefore, the mechanics of planning can be more precisely defined as choosing a mapping $b \in B$ (where B is the set of all possible mappings), from each $r^o \in R^o$ to a $g \in G_{r^o}$,

$$b : R^o \rightarrow G. \tag{7}$$

Given this mapping, the state of the robot evolves continuously,

$$\dot{x} = f(x, b(r^o_i)),\ \forall x \mid p(x) \in r^o_i. \tag{8}$$

Finally, the planning task can be summed up as choosing a mapping, b (Equation 7), from each region, r^o, to a control law g_{r^o} to be be executed in within that region

(Equation 8). The resulting hybrid execution must satisfy the acceptance condition, *Acc* (Equation 6) and minimize the cost incurred by the robot to minimize the cost of the execution of the trajectory,

$$b = \arg\min_{b \in B} \sum_{I \in \tau} c(r_I^o, M, b(r_I^o)) \mid Acc(\chi) = True. \tag{9}$$

This mapping is depicted in Figures 7 (using a uniform grid of regions) and 8 (using a non-uniform grid of regions).

Note the parallelism with the high-level characterization of the planning task in Equation 2. The only difference between Equation 9 and Equation 2 is segmentation of the plan over the hybrid execution , requiring the summation of the cost function over $I \in \tau$.

This representation of planning as an optimization over the design of a controlled hybrid automaton may seem straight forward, but there are dangers hidden in this model. Some components of the model may come from first-principle analysis, for example, the differential equations modeling the behavior of the robot. However, other components are based on systems that are too complex for this sort of analysis. The most important examples of these models of complex systems are the cost and transition models for the traversal of regions. The optimization relies heavily on these relatively coarse, discrete models of complex continuous systems. Therefore, it is to our benefit for the planner to have as good a model of the reactive execution of its plans as possible. Satisfyingly, this returns us to our research question in Chapter 1, how can conflicts between layers be detected and corrected in a hybrid deliberative-reactive robot control architecture, to improve system-level performance? In the next section we define a machine learning task to improve these models of the reactive layer's execution of the plan, based on measurements of actual executions.

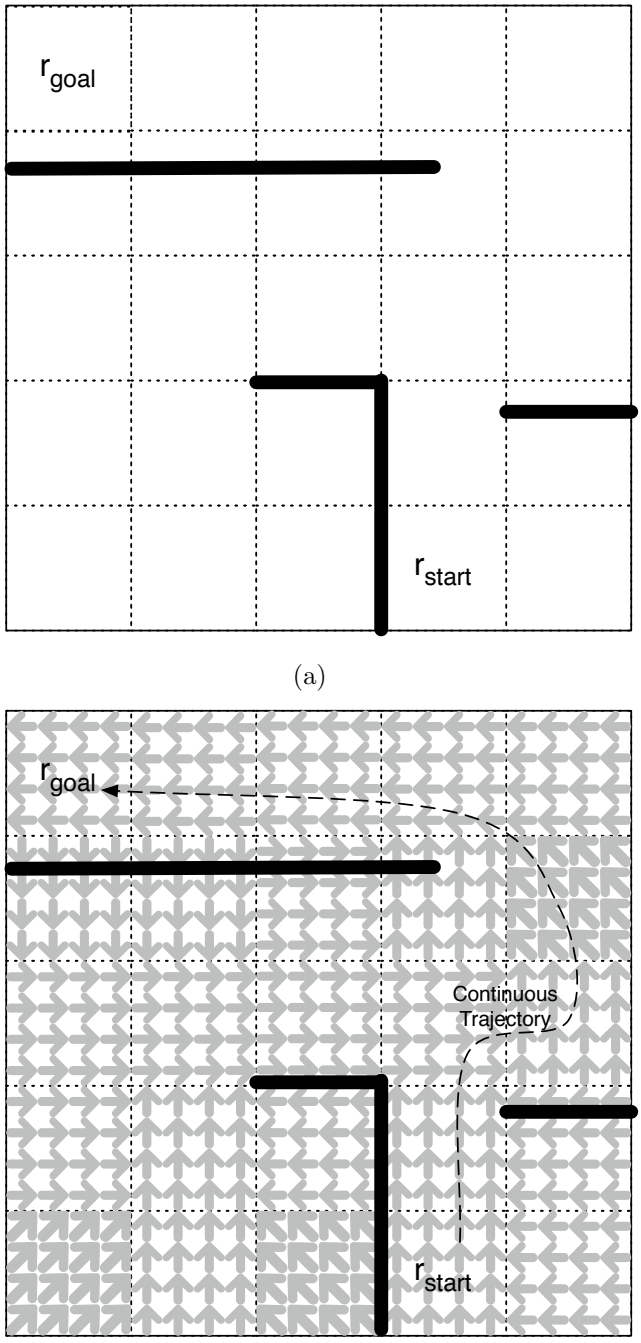
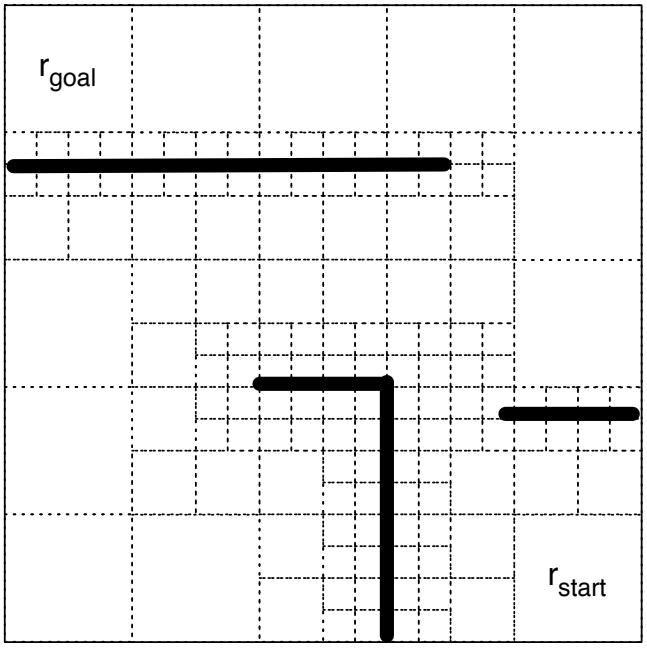
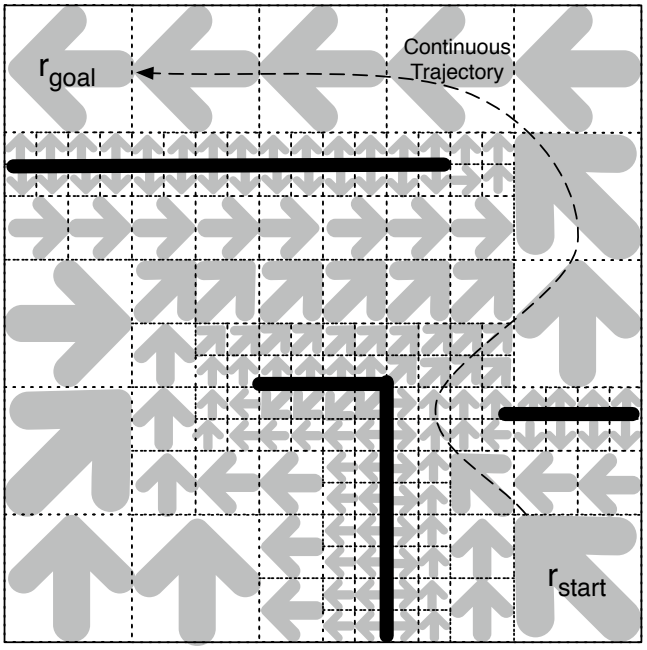


Figure 7: In (a) a possible world and map for the planning task. Obstacles in the map are shown as heavy black lines. In (b), the planning task is shown as choosing mapping from each discrete state (or region) to a controller to minimize the predicted cost of the robot's continuous trajectory. Consequently, as the planner's model of this cost improves, the cost of the resulting trajectory decreases.



(a)



(b)

Figure 8: Similar to Figure 7, in (a) a possible world and map for the planning task. Obstacles in the map are shown as heavy black lines. Different from Figure 7, the world is broken up into non-uniform regions in a manner similar to a quad grid. In (b), the planning task is again shown as choosing mapping from each discrete state (or region) to a controller to minimize the predicted cost of the robot's continuous trajectory.

4.3.3 Inter-Layer Conflict, Correction and Learning

Two components of the deliberative layer's planning process rely primarily on *a priori* models of the reactive layer's execution of the provided plans. The transition model, $d(r^o, M, g)$ predicts the the next region the system will enter, given the region the robot is currently in and the control law the robot is currently executing. The cost model $c(r^o, M, g)$ predicts the cost incurred by the system until the next region is reached. If the cost and transition models perfectly predicted the cost and state transitions of the continuous trajectory, finding an optimal discrete plan would result in an optimal continuous trajectory. As long as there is significant error in the cost and transition models, there is the potential for conflict between the deliberative model and the reactive execution.

It is the goal of this work to work toward better integration of the deliberative and reactive layers by learning a cost model that better represents the cost actually incurred by the reactive execution of the plan. Improving performance by learning the transition model as well will be discussed further in Chapter 8.

We begin by defining a measurement function, $h_c(x, r^o)$ which measures the cost incurred by the execution of control law g in region r^o given map M. Then, we define the conflict between the deliberative and reactive layers as the absolute difference between the deliberative cost model c and the external measurement of the reactive execution, h_c.

$$\mathit{Conflict} = |h_c(x, r^o) - c(r^o, M, g)| \tag{10}$$

It is this value that must be minimized to eliminate conflict between the deliberative and reactive layers.

We can then define the learning problem as choosing a cost model that best minimizes the inter-layer conflict, given all measurements up to time t,

$$c_t = \arg\min_{c \in C} \; E[\; |h_c(x, r^o) - c(r^o, M, g)| \;] \mid \{h_c(x, r^o)\}^t \tag{11}$$

where C is the set of all possible cost functions. This optimization is over a expectation not only because of possible noise in the sensory information, but because, as noted earlier, the mapping function is non-invertable. Therefore, the cost model, which is a function of M, cannot directly access the reactive output measured by the measurement function. The best the cost model can do is a prediction of the reactive output. Our goal is to minimize the error in this prediction, using the prediction to choose the mapping b,

$$b_t = \arg\min_{b \in B} \sum_{I \in \tau} c_t(r_I^o, M, b(r_I^o)) \mid Acc(x) = True,\ \{h_c(x, r^o)\}^t. \tag{12}$$

As noted with Equation 3, since the measurement function h_c cannot be known a priori, we must use the best available learned cost model, c_t, to approach the optimal mapping b^*,

$$b^* = \arg\min_{b \in B} \sum_{I \in \tau} h_c(r_I^o, M, b(r_I^o)) \mid Acc(x) = True \tag{13}$$

As the learned cost model becomes a better predictor of the measurement function, the planning function b_t will approach the optimal planning function b^*,

$$b* = \lim_{|h_c(x,r^o) - c(r^o,M,g)| \to 0} b_t \tag{14}$$

4.4 Implementation Guidelines

Given the above theoretical formulation, the question remains of how this formulation can be implemented in practice. While every implementation situation is different, certainly certain guidelines can be developed to aid in implementation. The following discussion covers some of the important implementation issues brought up by the above theoretical formulation.

4.4.1 Functional Layers

Because the goal of the above formulation is to correct conflict between the deliberative and reactive layers of a hybrid control system, it is likely that the existing

system is already logically divided into deliberative and reactive layers. If not, the system must be divided into logical components. In some systems, this division may be obvious. In others, the distinction may be blurred somewhat. For example, in the DAMN architecture [41], the reactive layer selects reactive control outputs from a discrete set of control outputs. While this component works over discrete choices and over a non-trivial spatial scale, it should still be considered a reactive layer since its output is executed as continuous control, and it does not deliberate toward a global goal.

4.4.2 Map

Since it serves as input to the learning component of the system, the map, M, is an important component from an implementation standpoint. The map need not be a human-readable geometric map, nor need it be a direct accumulation of sensor data. However, it is important that it relate a spatial location to a function of the sensor measurements collected at that location over time.

In general, the more direct the correlation between the map data and the sensor data it represents, the easier the learning task will be. The learning task is to learn a prediction of sensor-based reactive performance, given a map as input. The closer the correlation between the map data and the sensor data, the simpler the prediction function can be. For example, an occupancy grid would provide a direct relation to LIDAR sensors. The function relating the sensor input to the map representation is a form of binification. However, a map containing semantic classifications of household items would provide a very complex function to learn.

4.4.3 Regions

The division of the world into the set of regions, R, is an important process, because it defines the functional unit of not only the planner, but also the learner. The regions need not be similar in size and shape, but must be comparable under two functions.

First, the regions must be comparable under the cost model. This allows the planner to evaluate sequences of regions to navigate toward the global goal.

Second, the regions must be comparable under some distance function suitable for use in a learning algorithm. For example, given an occupancy grid as a map, evenly sized and shaped regions could be compared by vectorizing the underlying occupancy grid and computing the Euclidean distance between the vectors. Dissimilarly sized and shaped regions could still be compared using a function such as average density of the occupancy grid. However, this distance function would make the learning task more difficult, since it is reducing the amount of information going into the learner.

Given that the regions may be partitioned non-uniformly, it may be of value to choose the partitioning function intelligently. While it is difficult to generalize about the effect of the choice of partitioning on the planner, a partitioning that increases the distance between the regions, according to the distance function, may result in a function that is easier to learn.

4.4.4 Cost Function

The cost function, and thus the cost model, c, that predicts it, provides the value over which the planner is optimizing. We have defined the planner as an optimization over the cost model, subject to the constraints of the connectivity of the regions and the end-point condition of the goal region. Certainly, the cost function can be a function of more than one variable, but the variables must be weighted (in a possibly context-dependent manner) and combined into a single function. It is important to keep this in mind, as a system that uses travel distance as a cost function may not find solutions that are particularly good from the standpoint of time or fuel consumption.

The cost measurement function, h_c, must measure the same value the cost model predicts. The cost measurement is used as training input to the learning mechanism. An integration of the cost measurement over a region is paired with a representation

of the map inside that region to create a training instance. The learned cost model is a prediction of this measurement function, given a map inside the region. For best results, it is desirable that there is a strong correlation between the map data and the cost measurement. If there isn't, it will be difficult to learn a prediction of the cost.

4.4.5 Learning

The selection of an algorithm for the learning component is one of the most important decisions in the implementation of the system. However, it is also the most difficult to generalize about because it is dependent on the representation chosen for the map, the complexity of the cost function, the dynamics of the robot, and the nature of the environment itself. That said, a few guidelines can be offered.

Nearly any supervised learning algorithm is a candidate for implementation. Given a real-valued cost function, regression methods are an obvious choice. However, classification methods may also be used if discretization of the cost function is permissible. This discretization may be chosen *a priori* or using the training data as input.

Another consideration is the question of batch-trained algorithms versus lazy algorithms. Batch-trained algorithms (e.g. artificial neural networks) will run fast in execution, but are not easily updatable as new training examples are obtained. Lazy algorithms (e.g. k-nearest neighbors) are easily updatable on-the-fly, but run significantly slower in execution.

Finally, in general, the more information provided in the input to the learner, the more potential there is for accurate prediction of the cost function. However as more information and dimensions are added, more training examples will be needed to cover the input space.

4.5 Discussion

In this chapter, we have reviewed the notation used in expressing the hybrid automaton and demonstrated how a hybrid deliberative-reactive robot control architecture

can be represented in the same model. In developing this representation, we identified two components, the cost and transition models, that rely on *a priori* models of the continuous execution of discrete plans. We demonstrated that the integrity of the planning task, and thus the system-level performance, rely on the accuracy of these models. Finally, we proposed a machine learning problem to learn the cost model from actual continuous executions of discrete plans.

In later chapters, we will demonstrate this problem in implemented physical and simulated robotic systems. The notation developed here will provide the background for explaining the functionality these systems. It will also provide the foundation for experimenting with machine learning techniques to learn cost models for these systems.

CHAPTER V

EXPERIMENTAL PLATFORMS

5.1 Introduction

This work is largely a product of development on several mature robotic systems. The platforms outlined below are the platforms that have been used in demonstrating the concepts detailed in this work. These platforms were initially developed for different purposes, and, as is the case for most interesting robotic systems, are the result of the combined effort of many people. Because of the differences in purpose, the platforms make use of different sensor modalities, software components, and architectural patterns. One thing they do have in common is the use of hybrid control architectures to control their operation.

5.2 LAGR Platform

The LAGR robot platform was designed for the DARPA LAGR (Learning Applied to Ground Robots) program. The goal of this program was "to develop a new generation of learned perception and control algorithms for autonomous ground vehicles, and to integrate these learned algorithms with a highly capable robotic ground vehicle" [13]. The platform was designed to demonstrate learning from example [45] and learning from experience [29] within the context of navigation through unstructured outdoor environments. Frequent third-party testing within the scope of the program resulted in a robust, stable platform.

5.2.1 Hardware Platform

The LAGR robot, shown in Figure 9, uses two color stereo camera pairs, a front bump sensor, a Garmin GPS receiver, and an inertial measurement unit. The two stereo

Figure 9: The LAGR hardware platform.

camera pairs each provide stereo depth maps at a range of 6-10 meters. The robot's turning axis is located at the center of the front axle, with the rear unpowered wheels turning on casters. It is approximately 90 centimeters long, 60 centimeters wide and 60 centimeters tall. It weighs approximately 90 kilograms.

Four Linux-based computers provide the robot's computing power. Two computers are devoted to perceptual processing. The bulk of this computation is devoted to computing stereo depth maps and appearance-based learning [54]. Mapping, control and planning processes are run on a single computer. The final computer performs lower-level functions (e.g. motor control, GPS/IMU integration).

5.2.2 Software Platform

Sensing on the LAGR platform relies heavily on the two stereo camera pairs mounted on its front mast. Each pair of cameras collect color images from which stereo depth maps are calculated. Using the known intrinsics and extrinsics of the cameras, the stereo depth maps are projected into a local height map. This height map is then

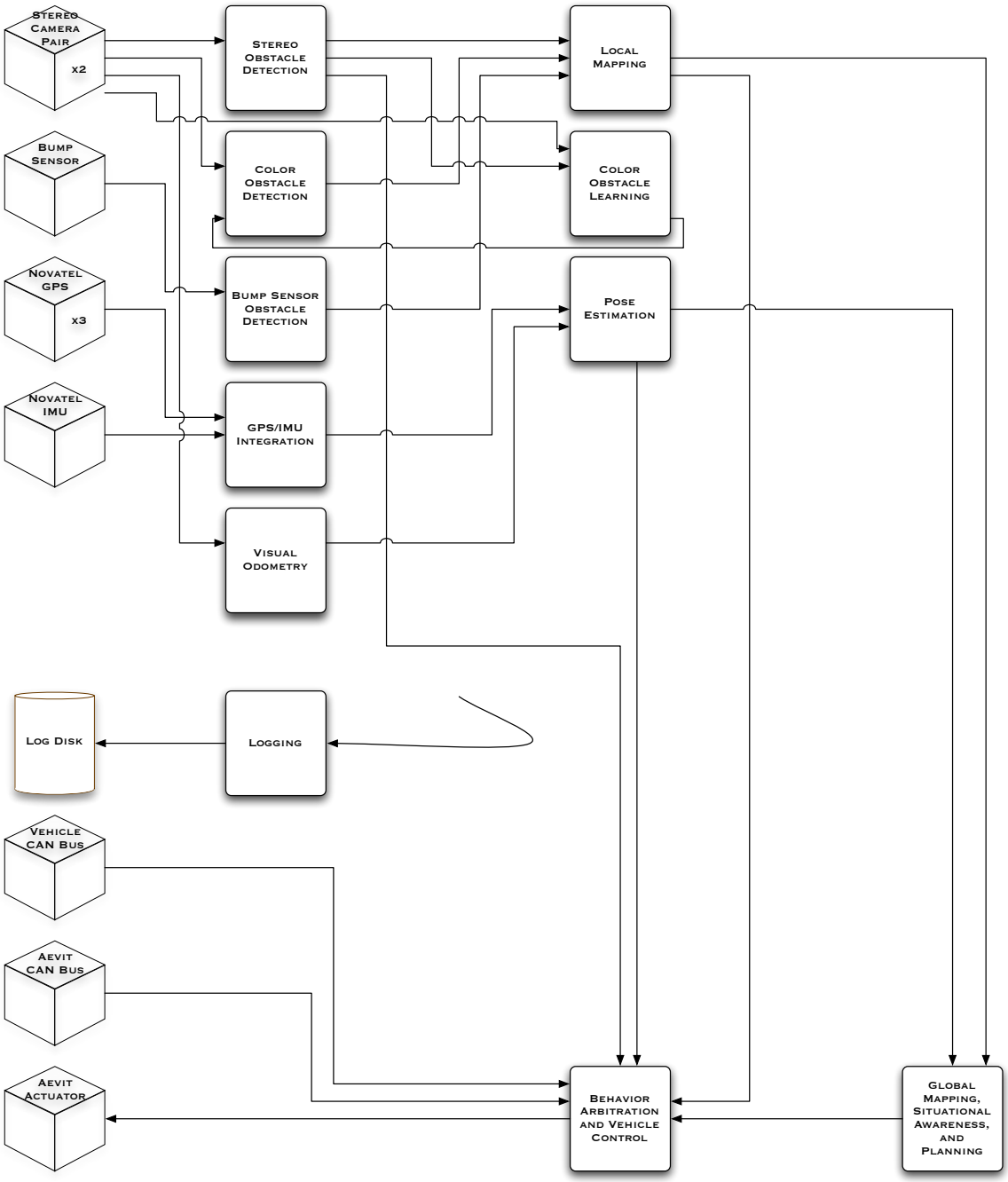


Figure 10: The Georgia Tech LAGR software architecture.

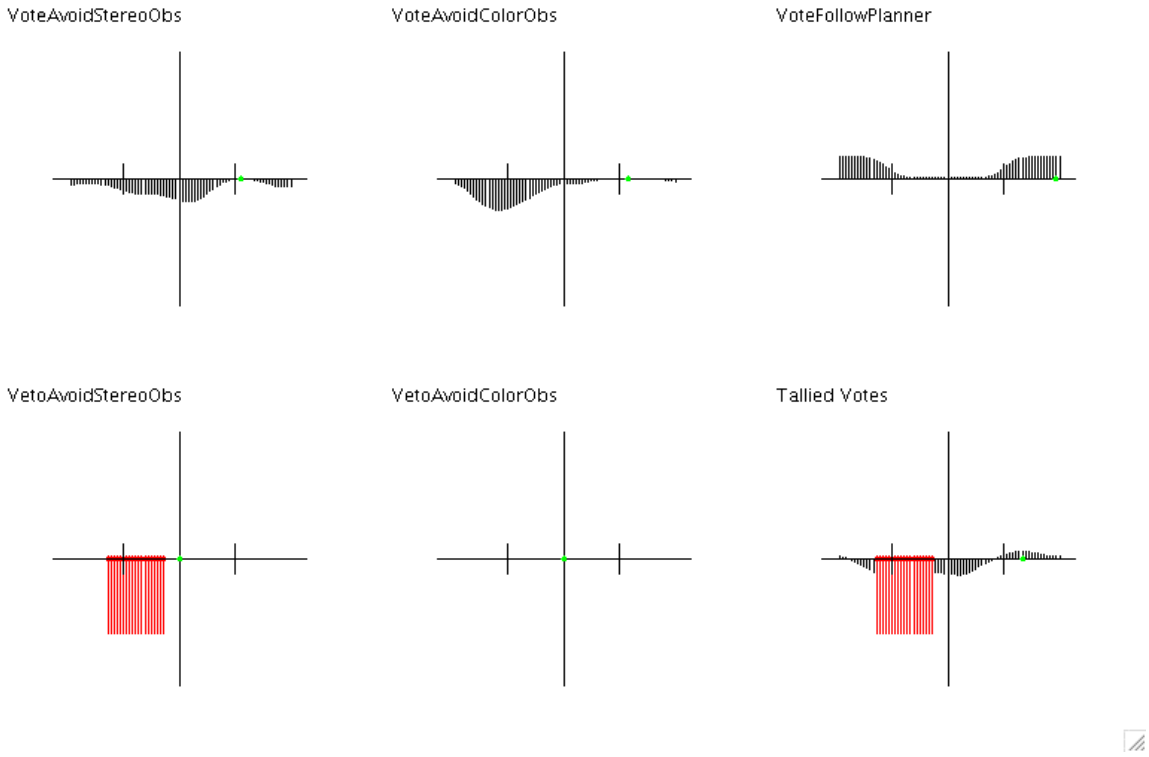


Figure 11: A graphical representation of the voting scheme employed to navigate the robot. The x-axis of each plot represents an ego-centric angular distribution of possible paths around the robot in the range $(-\pi, +\pi]$, with 0 being in front of the robot. The y-axis represents the relative preference of each path, according to the respective controller. Vetoes are drawn as large negative values. The last plot represents the sum of the votes provided by all the controllers. The largest non-vetoed value is chosen for action by the robot. In this example, the first behavior resists a stereo-perceived obstacles to the front and left of the robot. A color-based obstacle is perceived to the left. The plan tells the robot to go backwards, and the left is vetoed as a result of stereo obstacles. The tallied votes tell the robot to go to the right.

converted into a height derivative map, using a Sobel filter, before being passed on to the mapping process.

The color information in the images, in combination with the local height derivative map, is used to train a learning algorithm to classify preferable terrain based on color, outside of stereo range. The local height derivative map is used as a supervisory signal. The color images are projected onto the same local map, and a function is learned, relating color to expected height derivative. This function is then used to classify terrain outside of stereo range as either preferable or not preferable. This classification is projected into a local map, and passed on to the mapping process.

The mapping process receives the streams of information from each camera and the front bump switch, and projects it into a global map, based on the robot's global pose estimate. Both the relatively high-accuracy, short-range stereo-based information, and the low-accuracy, long-range preferability information are integrated into the map. The map is based on a grid construction, with a resolution of 0.1 meters. New information is combined with old using an aggressive replacement policy, favoring newer information over older information.

The LAGR control architecture, shown in Figure 10, uses a three-layer, behavior-based control approach. This approach makes use of three functional layers to control the robot: a deliberative layer, and sequential layer, and a reactive layer.

The deliberative layer consists of a global planner that reasons over the global map. The global planner balances the value of each type of information to produce global navigation plans, expressed to the reactive layer as a series of waypoints through the environment. Both an implementation of A* and the combinatorial planner described in [55] can be used by the global planner, dependent on parameterization.

The sequential layer is implemented as a finite state automata (FSA) that maintains a discrete representation of the robot's state over time. Transitions within the FSA are driven by triggers based on the perception modules. Each state within

the FSA represents a characterization of the robot's state in the environment and corresponds to a specific mode of operation for the reactive layer.

The reactive layer is made up of primitive behaviors that express their preferences by voting, in a manner similar to [41], [42]. Each behavior communicates its particular interests by distributing an allocation of "votes" either for or against candidate straight-line paths at headings in five-degree increments from the robot's current position. The behavioral arbiter chooses a direction in which to travel by summing the votes for each heading, according a predetermined weighting scheme, and choosing the heading with the largest total. In addition to voting for or against headings, each behavior has the opportunity to "veto" any heading it deems unsuitable. This veto can be interpreted as allocating an infinite vote against the vetoed heading. The arbitrator respects these vetoes by selecting for execution the heading with the largest vote total that has not been vetoed by any behavior.

The deliberative layer's output is interpreted by the reactive layer as a series of waypoints to be achieved, one after another. A single behavior expresses this interest by always distributing votes among headings that bring the robot closer to the next waypoint. This behavior is, of course, used in conjunction with other behaviors to create a behavioral assemblage that follows the global plan while smoothly navigating locally favorable terrain.

The sequential layer's output is interpreted by the reactive layer as a selection of weights to be used by the behavioral arbitrator in summing the votes expressed by the behaviors. Each state in the sequential layer's FSA corresponds to a unique mapping of behaviors to weights. For any state in the FSA, some weights may be zero, effectively removing the mapped behavior from the current assemblage. For example, when the robot's bump switch is triggered, the sequential layer's FSA transitions into a "back-up" state. In the weighting scheme mapped to by the the "back-up" state, the "move-to-waypoint" behavior has a zero weight.

Because of extensive experimentation and third-party testing, the LAGR platform is a robust and capable platform for robot navigation experiments. It is capable of relatively long navigational runs in complex, unstructured environments.

5.3 Sting Platform

The Sting robot platform was designed and fielded as an entry to the DARPA Urban Challenge [14], developed in cooperation by Georgia Tech and SAIC. The Urban Challenge was a DARPA-sponsored competition to advance the state of the art in autonomous vehicles, particularly, "vehicles that safely execute missions in a complex urban environment with moving traffic". Competing robots were tasked with completing a 60-mile navigation course through challenging urban terrain, interacting with other vehicles, both manned and unmanned. Robots were judged not only time to complete the mission, but also by qualitative measures, including safety, legality within a real urban environment, and predictability by human driver standards. With this goal in mind, the Sting platform was designed for autonomous navigation in relatively structured, dynamic environments [56].

5.3.1 Hardware Platform

The base hardware platform is a 2006 Porsche Cayenne SUV. It is retrofitted with a commercial drive-by-wire system, allowing electronic control of the steering wheel, accelerator, brakes, transmission, and external signals. Eight dual-core Linux-based computers, connected by Gigabit Ethernet, provide computing power for the robot.

Perception is provided in several modalities by a variety of sensors. Static obstacle detection is provided by six planar laser range finders oriented in the x-y plane and four more in a "push-broom" configuration on the vehicle's roof. Dynamic obstacles are detected by four Doppler radar heads. Six color cameras provide paved and unpaved road detection. Integration of a GPS signal and an IMU provide pose and velocity estimation.

5.3.2 Simulation Platform

Figure 12: The Sting hardware platform.

A simulation analog to the hardware platform was also constructed within the popular Gazebo simulation environment [24], as shown in Figure 13. This simulation was used in testing and verifying the software as it was developed. The simulated platform matched the hardware platform's physical dimensions and kinematic constraints. It also accurately simulated the hardware platform's sensing abilities with respect to static and dynamic obstacles. It did not, however, accurately reproduce the vision-based sensing modalities of the hardware platform. Therefore, functionality that relied on vision-based perception had to be tested exclusively on the hardware platform.

5.3.3 Software Platform

Sensing on the Sting platform follows a multi-modal approach. Respective hardware sensors are devoted to detecting different classes of environmental features. Static

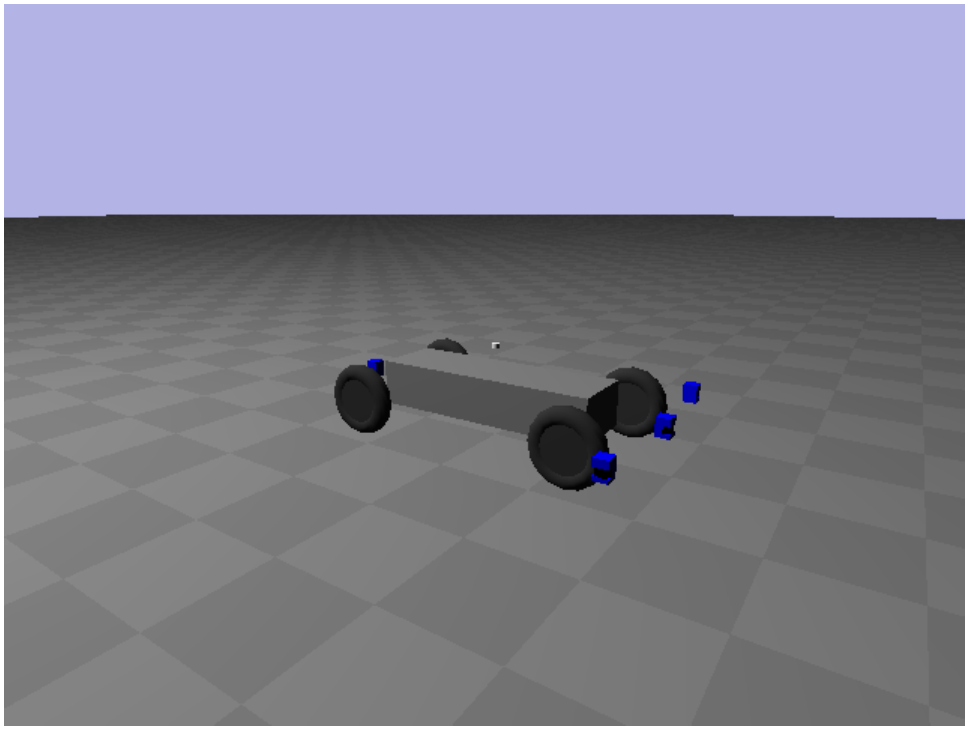


Figure 13: The Sting simulation platform.

obstacle detection is performed by integrating data from the ten laser range sensors mounted around the vehicle. Range and bearing information is projected into the local frame, clustered into distinct objects, and classified according to size and shape. Dynamic obstacle detection is performed by integrating the static obstacle detection with the range and velocity information provided by the four radar heads. Paved roads are detected and characterized by a template-based vision algorithm [30]. Unpaved roads are detected and characterized by a statistical inference-based vision algorithm [26].

Mapping follows a two-layered approach. At the higher-level, a graph-based a priori map is used to define the road network on which the vehicle's mission will be executed. Roads are represented as edges between geo-located vertices. Vertices are sparse enough that sensing beyond GPS is required to successfully navigate curved roads. At the lower-level, a gird-based metric occupancy grid is used for local spatial

planning. Sensor data is integrated into the global frame, based on the vehicle's estimated pose.

The control architecture follows a hierarchical, hybrid approach, but is more complex than the three tier architecture of the LAGR platform. Notable is the addition of a second deliberative layer, concerned with strategic symbolic planning, working at a higher-level than the local navigation planner, as shown in Figure 14. The mission-level strategic planner provides a symbolic plan over the high-level graph-based map, recommending a mode of operation (such as drive along paved road, drive along unpaved road, drive through intersection, or drive in unstructured area) along each edge in the graph. This plan is passed on to the tactical deliberative layer and the sequential layer in the form of a commanded task.

The tactical deliberative layer maps the commanded task to a spatial planning goal. For some tasks (e.g. "drive to a point in an unstructured environment"), the goal is a location in the global map. For other tasks (e.g. "follow paved road"), the goal is to navigate along a perception-based signal (i.e. the detected road) as far as the horizon of usable perception.

The sequential layer builds upon the paradigm of the finite state automata, implementing a "nested" finite state automata [56]. In a nested finite state automata, a top-level automata maintains the high-level state of the robot. Corresponding to each state in the automata, is another sub-automata which maintains a state as long a its parent is the current state at its respective level. The advantage over a standard FSA is a compactness of representation and the ability to express asynchronous transitions between layers. Input from the strategic deliberative layer is interpreted as perceptual input to the transitions in the top level automata.

Figure 15 relates the nested finite state automata structure to urban driving. Within the "follow lanes" state is nested an automata that steps through the process of driving in a lane based on perceptual signals ("follow lane"), falling back on

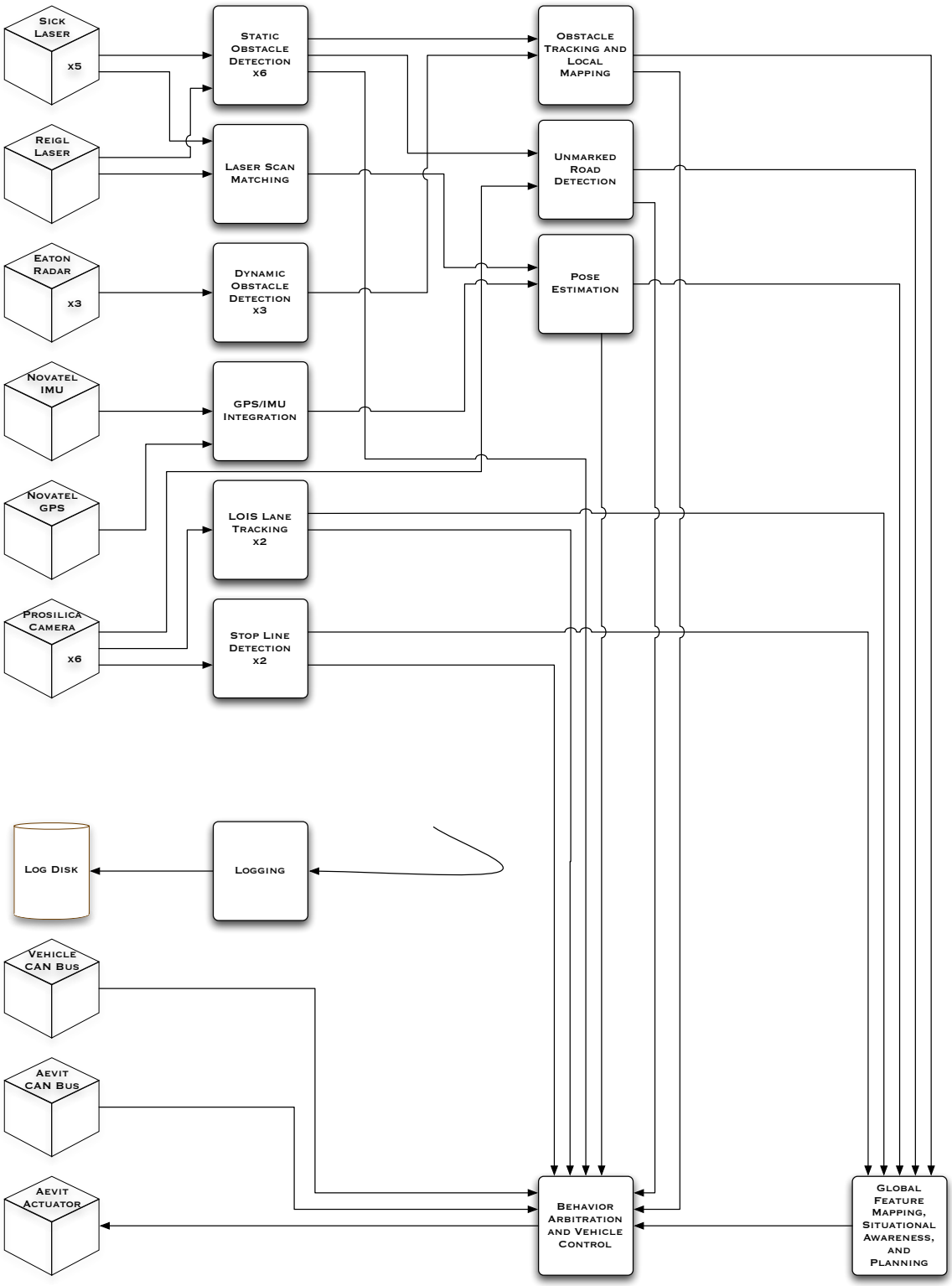


Figure 14: The Sting software architecture.

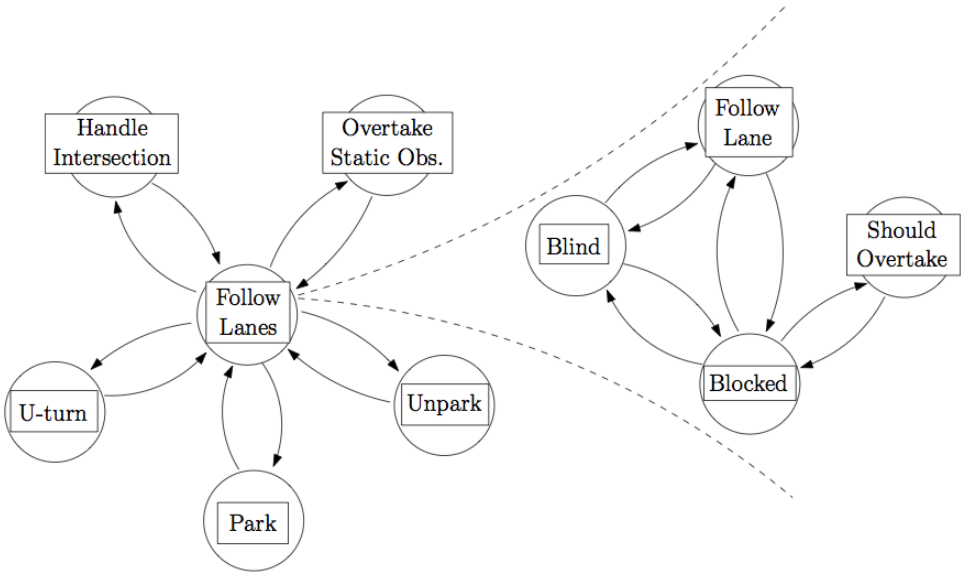


Figure 15: Nested Hybrid Automata.

ladar/map-based navigation when perceptual signals fail ("blind"), pausing behind another vehicle stopped in the lane ("blocked") and overtaking said vehicle ("should overtake").

The reactive layer is implemented by a voting scheme similar to the one described for the LAGR platform. Instead of voting over straight-line paths, the behaviors vote over circular arc paths at increments of curvature from the robot's current position. As with the LAGR platform, the state of the sequential layer is mapped to a weighting scheme over the voting behaviors, and plan produced by the tactical deliberative layer is interpreted as a series of goal locations by one or more of the behaviors.

In its run to the Urban Challenge semi-final round, the Sting platform demonstrated many advanced navigation capabilities, including driving autonomously at 35 mph in the presence of other traffic. The successful development of the platform was due, in part, to an expressive, flexible architecture, and developmental testing within a realistic simulation environment.

5.4 Discussion

These robotic systems represent the experimental platforms used in the work discussed in this document. Both platforms represent significant development and testing by many people. Because of this attention to performance, they are robust and reliable platforms for testing. Additionally, each implements an example of a hybrid deliberative-reactive control architecture. Because of this, each lends its self to the approach of this work well.

CHAPTER VI

ACTING ON DETECTED INTER-LAYER CONFLICT

6.1 Introduction

As described in Chapter 3, hybrid control architectures provide flexibility along four axes: temporal scale, temporal resolution, spatial scale, and spatial resolution. The reactive layer must operate with a high temporal and spatial resolution in order to afford guarantees that the robot is kept in a safe and allowable state. By decoupling the controllers from the mapping and planning processes, the deliberative layer is afforded more flexibility to deal with larger temporal and spatial scale, in terms of cycle regularity and frame rate.

Many implementations of the deliberative layer include a global planner working over a planar map, and considering a 2 degree-of-freedom description of the environment and the robot. This simplification of the state space allows the global planner to manage the complexity of planning long paths through complex environments.

The reactive controllers (which operate on a smaller spatial scale), however, can take the full kinematics and dynamics of the robot into consideration. For example, yaw, pitch and roll can be considered together with position, resulting in a higher dimensional configuration space. If the global planner often can barely keep up with real-time constraints in a two degree-of-freedom problem, attempts to plan paths through the full configuration space would likely be infeasible.

It is within these constraints that we propose a first attempt to detect the conflict between the deliberative and reactive layers, and correct this conflict within the deliberative layer. A trivial learning mechanism is implemented by allowing the deliberative layer to project higher-dimensional "obstacles" detected at the reactive

controller-level into the planar global map [57]. This system is implemented on the LAGR robot platform, and comparatively demonstrated in a complex outdoor environment.

6.2 Approach

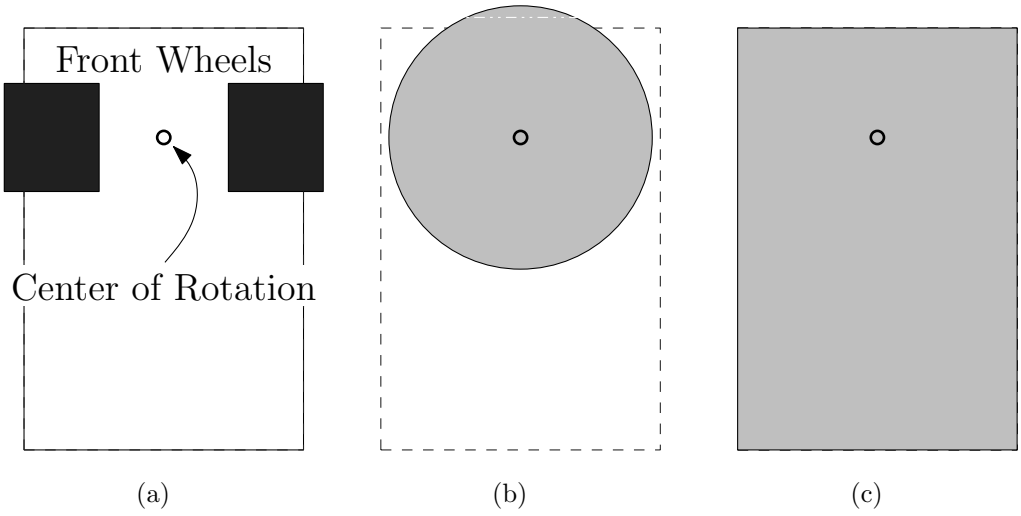


Figure 16: Footprints used for the LAGR robot. (a) A diagram of the robot (pointed up) and its center of rotation. (b) The "optimistic" footprint used by the planning process. (c) The "pessimistic"/accurate footprint used by the low-level controllers.

To reduce the computational complexity of planning a global navigation path, many global planners use a reduced state space representation of the robot. For example, the planner might use a two degree of freedom (x, y) representation when the robot's full state space is represented by a three degree of freedom (x, y, θ) representation. Figure 16 illustrates this sort of simplification. Figure 16(a) depicts the physical structure of the LAGR robot (described in detail in Chapter 5). Figure 16(b) depicts the two degree of freedom simplification of the robot, idealizing the robot as a circle with a diameter equal to the width of the robot at the drive wheels. This simplification of the configuration space is valid as long as the robot drives straight ahead, but underestimates the size of the robot as it turns, especially as it turns in

place. In fact, because of the elimination of θ from the state space, the rotational kinematics of the robot are are not taken into account at all. In effect, the deliberative layer is being "optimistic" about the robot's capabilities. Because the reactive layer is working over a smaller spatial and temporal scale, it can afford to employ the full configuration space (x, y, θ), as shown in Figure 16(c). This representation reflects a more accurate model of the robot's capabilities.

Certainly, an argument could be made for using a "pessimistic" over-estimate of the robot's footprint (perhaps a circle circumscribing the robot's actual footprint) in the deliberative layer. However, potential problems with that approach emerge. For example, as mentioned in Chapter 3, it is possible that no acceptable solution exists for a pessimistic planner, when an accurate representation of the robot's kinematics would find a solution.

This work aims to address this problem of reconciling conflicting representations of the robot's capabilities between architectural layers. In this implementation of the approach described in Chapters 3 and 4, we implement the deliberative layer with a global path planner to continually re-plan based on updated sensory input from an otherwise unknown environment. The global path planner makes use of a simple two degree of freedom optimistic model of the robot. The reactive controllers carry out the global planners plan, operating on an accurate model of the robot and the local environment, and a full three degree of freedom configuration space. When a conflict occurs between the two models of the robot, the constraints, as detected in the more accurate model in the reactive layer, are projected into the smaller state space of the deliberative layer, preventing extended conflict within the system.

6.3 Mapping, Planning and Control

In this section, we briefly describe the experimental robot platform, system integration, and planning and control tools, relating them to the notation developed in

Chapter 4. Within this chapter, the LAGR robot platform was used in development and experimentation. Further description of this system is provided in Chapter 5, [45] and [55].

6.3.1 Map Processing

The LAGR platform implements mapping over a grid-based terrain derivative map. The LAGR robot's two vision processing computers collect camera images, compute stereo depth maps, and, using the known camera intrinsics and extrinsics, project them into a local height map. These height maps are transformed into a height derivative map and passed on to the mapping process where they are integrated into the global frame, according to the robot's estimated global position. The vision processes also use the camera images to compute estimates of preferability of points in the local frame, as described in Chapter 5 and [29]. This information is also integrated into the global map by the planning process. This data is stored in grid cells of fixed size, with a resolution of 0.1 m. New information is combined with information already stored in the map according to an aggressive replacement strategy.

Relating the map to the representation developed in Chapter 4, the world, W is implemented as a 200×200 meter area in the x-y plane. The map, M is implemented as a height derivative map with square grid cells at a .1 meter resolution. The set of regions, R is implemented as a grid superimposed on the map, at a .3 meter resolution, corresponding to the size of the two dimensional configuration space used by the planner.

6.3.2 Path Planning

Spatial path planning is executed over the global map described above. The system is capable of using either an A* path planner, or the combinatorial planner we described in [55]. For the purposes of this discussion, either planner may be used, as they both provide input to the reactive layer by performing the optimization defined in Equation

9,

$$b_t = \arg\min_{b \in B} \sum_{I \in \tau} c(r_I^o, M, b(r_I^o)) \mid Acc(\chi) = True.$$

albeit over different representations. Recalling from Chapter 4, the function b maps each interior region r^o to a controller, g. In this case, the planner maps each region to a next waypoint, used to parameterize the reactive controllers, selecting a controller $g \in G_{r^o}$. The transition model for each controller, $d(r^o, M, g_{r^o})$ returns the neighboring region associated with the next waypoint.

Both the stereo and color vision-based data may be used in calculating the cost model c. In this particular experiment, the cost model is implemented as a piece-wise linear function over just the derivative map,

$$c_0(r^o, M, g) = \begin{cases} 1 & if\ \ avg(M_t(x)\ \forall x \in r^o) < d_{low} \\ 1 + k \cdot avg(M_t(x)\ \forall x \in r^o) & if\ \ d_{low} < avg(M_t(x)\ \forall x \in r^o) < d_{high} \\ \infty & else \end{cases}$$

where $M_t(x)$ is the value in the derivative map at x, d_{low} is a constant representing a value in the derivative map that is always traversable, d_{high} is a constant representing a value in the derivative map that is always intraversable, k is a scaling constant, and $avg(M_t(x),\ \forall x \in r^o)$ is the average derivative for all values in the map in r^o. The planner passes on the next waypoints to the reactive controller, which works to achieve them within its local scope.

6.3.3 Motion Control

The robot is modeled as moving through the world in in the x-y plane, and having an orientation,

$$x = \begin{Bmatrix} x \\ y \\ \theta \end{Bmatrix}$$

The robot's dynamics are modeled in this space using the robot's rotational and translational velocities,

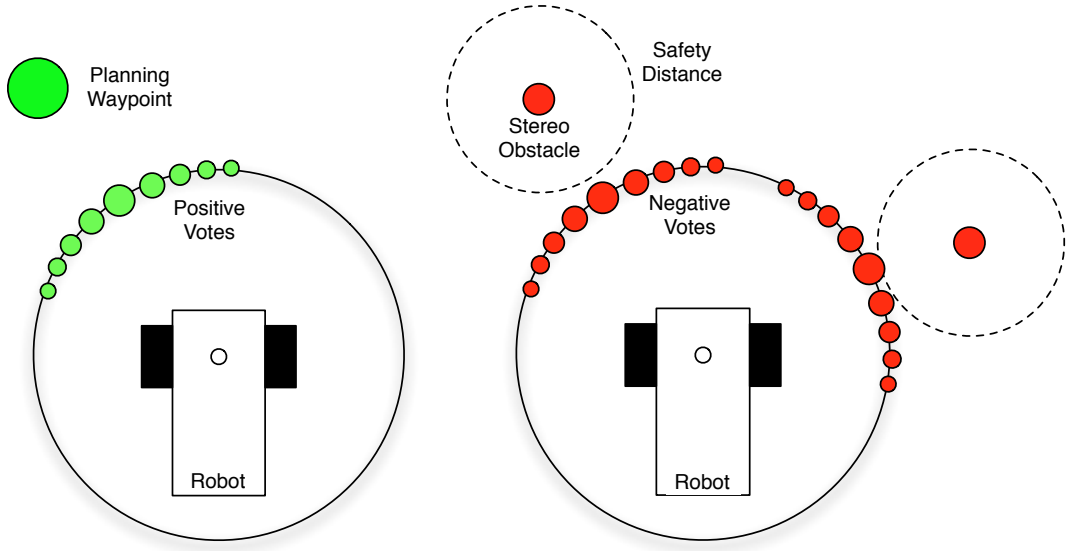
$$f(x) = \left\{ \begin{array}{l} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ \omega \end{array} \right\}$$

The robot's translational and rotational velocities are modeled as uncoupled because the robot's drive mechanism is a differential drive transmission. The robot can achieve arbitrary combinations of translational and rotational velocities, subject only to maximum wheel speeds. Correspondingly, the controller outputs rotational and translational velocities,

$$g(x) = \left\{ \begin{array}{l} v \\ \omega \end{array} \right\}.$$

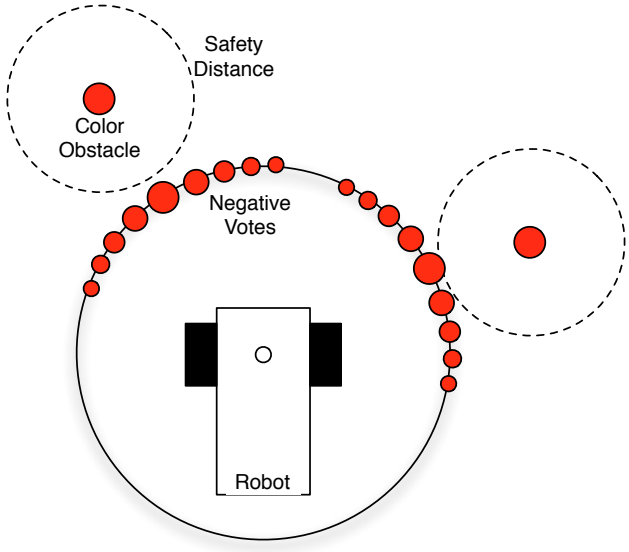
Reactive control is implemented in a manner inspired by the DAMN [41] architecture, as mentioned in Chapter 5. In this implementation, individual controllers, representing specific interests related to the robot's overall objective, are given an allotment of "votes" which they may cast for or against actions that will work to achieve their goals. An arbiter sums the votes, choosing the action with the highest tally. Similar implementations have been successfully deployed in several robotic navigation tasks [42], [53].

In this implementation, the actions evaluated are straight-line paths from the robot, with headings at a resolution of 5 degrees around the robot. The controllers take the kinematic constraints of the robot into account by evaluating the effect of first turning to the desired direction (effecting the rotational component of the motor command) and then traveling in that direction (effecting the translational component of the motor command). This turns out to be a reasonable approximation of the robot's low-level controller, which implements a relatively aggressive rotational gain, prescribing a translational and rotational velocity, given the desired angle.



(a) The *move to waypoint* behavior.

(b) The *avoid stereo obstacles* behavior.



(c) The *avoid color obstacles* behavior.

Figure 17: The three voting behaviors used together in in the reactive layer.

In this experiment, the following voting controllers are used, diagrammed in Figure 17:

move-to-waypoint - casts positive votes in the direction of the next point on a list of waypoints provided by the planner. Votes are distributed according to a Gaussian function centered on the direction of next waypoint.

avoid-stereo-obstacles - casts negative votes in the direction of any obstacles sensed by the stereo vision system. Votes are distributed according to a sum of Gaussian functions, each centered on a sensed obstacle.

avoid-color-obstacles - casts negative votes in the direction of any obstacles sensed by the traversability (i.e. color-based) vision system. Votes are distributed according to a sum of Gaussian functions, each centered on a sensed obstacle.

One potential pitfall of arbitration over votes is misallocation of each controller's allotment of votes. If the controllers' voting weights are not properly balanced, one controller may dominate the arbitration, either preventing the robot from making progress to higher-level goals or allowing the robot into undesirable states. Because this weighting is typically an empirical process and dependent on implementation and environment, we have added robustness in a manner similar to [44] by supplementing the voting scheme with "vetoes". Each controller, in addition to its allotment of votes is given the option to veto each of the available actions. The arbitrator respects the vetoes by ignoring actions that have been vetoed by at least one controller, regardless of how many votes those actions have received.

The strategy is that actions which are deemed to put the robot in imminent danger should be vetoed. What qualifies as "imminent danger" must be decided on a controller-by-controller basis. Because the burden is only to identify dangerous paths over a short distance, the full dynamics of the robot can be considered, including

collision checking of rotations necessary to achieve the desired orientation and the feasibility of various maneuvers given the slope of the terrain.

Like the voting controllers, these vetoing controllers use a set of straight-line paths with headings at a 5-degree resolution as the action set, and are diagrammed in Figure 18:

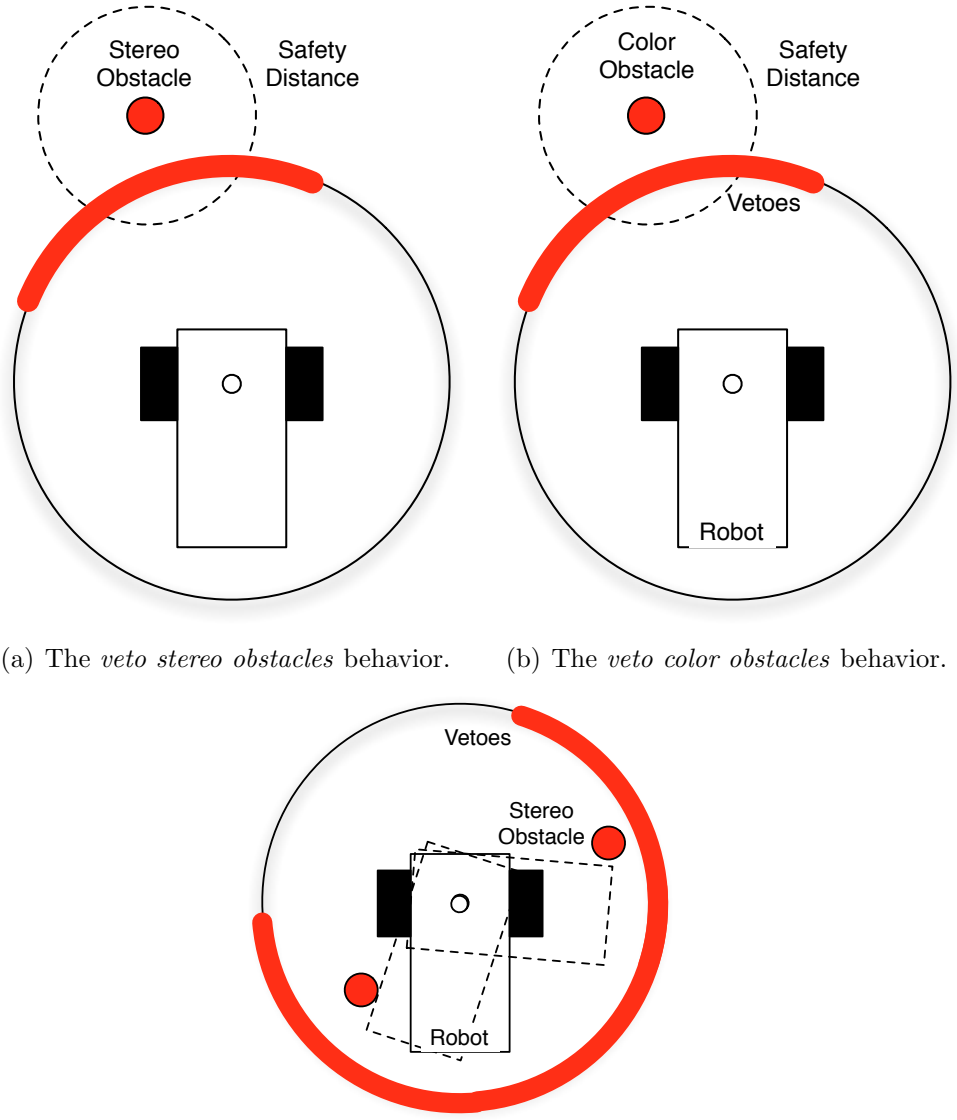
veto-stereo-obstacles - casts vetoes against any direction which will bring the robot into a collision with a stereo vision-sensed obstacle within one meter of the robot's current position, taking into account the robot's configuration space.

veto-color-obstacles - casts vetoes against any direction which will bring the robot into a collision with a color vision-sensed obstacle within one meter of the robot's current position, taking into account the robot's configuration space.

veto-dangerous-turns - casts vetoes against any direction which cannot be achieved by turning in place, due to obstacles blocking the path of robot over the proposed rotation.

Following the approach described in Chapters 3 and 4, we implement a mechanism by which the deliberative layer is (in part) informed and driven by the reactive layer. When conflict between the layers, with respect to the representation of the robot's capabilities, is detected, the deliberative layer corrects that conflict within its own representation. That is, the reactive layer's higher-dimensional configuration space is accounted for by the deliberative layer by projecting detected inter-layer conflicts into the deliberative layer's state space.

Figure 19 depicts a situation in which a conflict between layers might be detected. Two obstacles leave a small opening, allowing a feasible path to pass between, given that the global planner assumes some relatively optimistic configuration space for the robot. However, the reactive controllers, which consider higher dimensional kinematic



(a) The *veto stereo obstacles* behavior.

(b) The *veto color obstacles* behavior.

(c) The *veto dangerous obstacles* behavior.

Figure 18: The three voting behaviors used together in in the reactive layer.

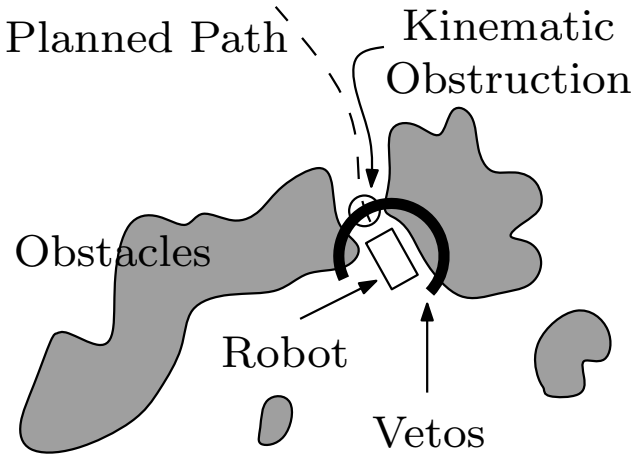


Figure 19: Illustration of basic operation of the proposed framework.

and dynamical constraints do not allow this action, vetoing the action. In the proposed system, a correction mechanism detects this conflict and inform the deliberative layer that the point (highlighted in the figure) should be marked as intraversable.

Within the format of the of the experience structure proposed in Chapter 3, we propose the following implementation of a learning experience:

command - the next planning waypoint

end criteria - transitioning into a neighboring region

context - the position in the map robot is trying to navigate to achieve the next planning waypoint

measured outcome - whether the reactive layer vetoed the direction of the next waypoint or not

Turning to the representation developed in Chapter 4, this can be viewed as adapting the planner's cost function (albeit in a simple manner) according to detected conflicts between the deliberative and reactive models of the robot's capabilities.

Conflict is detected when a behavior in the reactive layer must veto the direction of the next planning waypoint. As conflicts are detected, the way-points at which they occur are stored in a set, $P_{conflict}$. As the planner continues to execute over the map, it uses the an updated cost function, c_t,

$$c_t(r^o, M, g) = \begin{cases} \infty & if\ x \in P_{conflict}\ \forall x \in r^o \\ c_0(r^o, M, g) & else \end{cases}$$

This implements a trivial learning algorithm, remembering cases that have already been experienced, and the value of the corresponding learned concept. Given this implementation of c_t, the planner is able to select a plan, as defined in Equation 12,

$$b_t = \arg\min_{b \in B} \sum_{I \in \tau} c_t(r_I^o, M, b(r_I^o)) \mid Acc(x) = True,\ \{h_c(x, r^o)\}^t.$$

6.4 Experiments

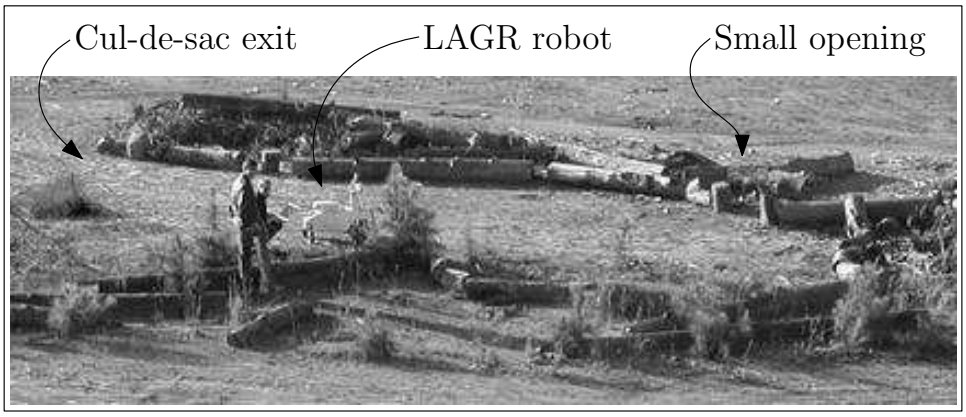


Figure 20: An overview of the experiment site.

In order to highlight the benefit associated with the proposed system and to illustrate its practical usefulness, a series of controlled experiments was performed in an outdoor terrain populated by small pine trees, fallen logs and other vegetative obstacles, as seen in Figure 20. In the following subsections, we summarize the outcomes of these experiments and highlight the differences in performance.

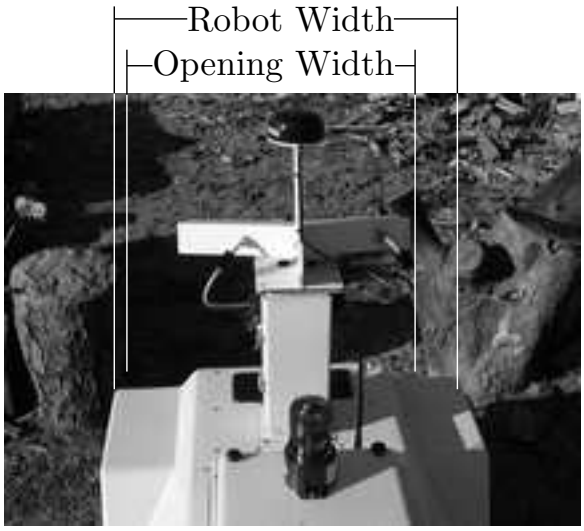
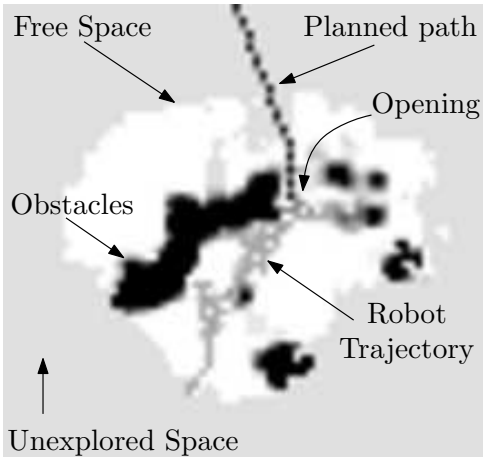


Figure 21: Opening in the Cul-de-sac.

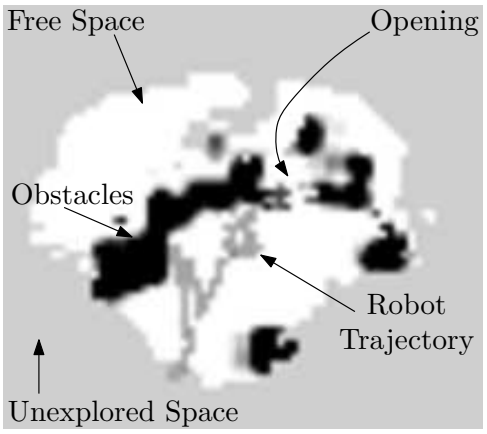
The experimental setup considered was the following: The robot started in the interior of a cul-de-sac with one small opening along one of its walls. This opening was wide enough for the optimistic planner to find a feasible path through the opening. However, the reactive controllers will find the opening to be too narrow for safe passage, and as a result, they will veto any attempt to drive through it. For each experiment, the robot was started with no a priori information about the environment except its relative position to the global goal. Trials were run using only a global planner, only reactive controllers, a global planner in conjunction with reactive controllers, and a global planner in conjunction with reactive controllers and the experimental conflict detection mechanism.

6.4.1 Global Planner Only

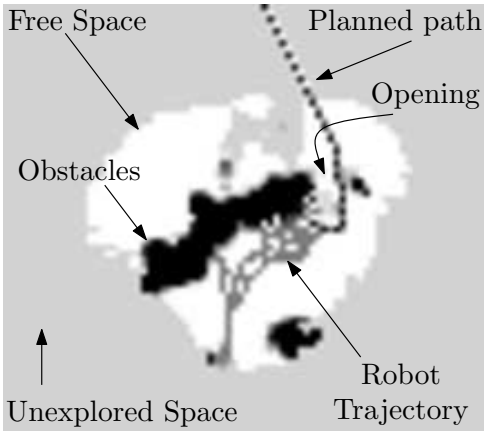
In the first run, only the planner was affecting the motion of the robot, and the only active low-level controller was a plan-following controller. As was to be expected, the planner found the opening and tried to push through (Figure 22(a)), with the result that the robot crashed into one of the logs defining the boundary of the opening (Figure 21). It should be noted that this problem can be remedied by making the



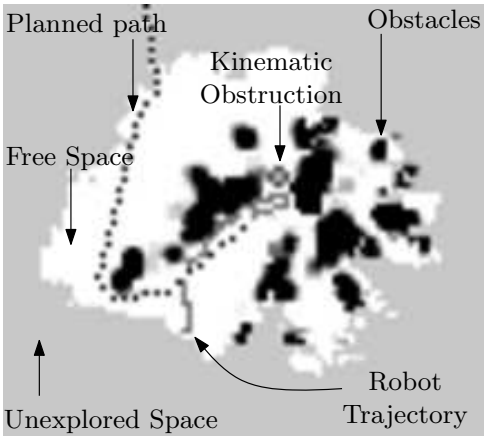
(a) The map, trajectory and plan resulting from an experiment using only a planner. Because the planner is too optimistic for the configuration space of the physical robot, the robot collides with obstacles while trying to navigate through the narrow opening.



(b) The map and trajectory resulting from an experiment using only reactive controllers. The safety-minded controllers kept the robot a safe distance from all obstacles, but did not allow progress to the goal location.



(c) The map, trajectory and plan resulting from an experiment using a planner which influences reactive controllers. The optimistic planner guides the robot toward the narrow opening, while the safety-minded controllers prevent the robot from entering. The result is that the robot loiters around the mouth of the opening.



(d) The map, trajectory and plan resulting from an experiment using a planner which influences reactive controllers with a correction signal back to the deliberative layer. The planner initially guides the robot toward the narrow opening, but the reactive controllers veto this action, noting that action in the global map. Using this information, the planner finds a path through the only safe opening in the Culde-sac.

Figure 22: Trajectories and maps created by the four experimental systems.

planner less aggressive and allowing for a larger, explicit safety-footprint. However, one of the basic ideas behind the system framework is to let the planner be aggressive and optimistic, and let the reactive low-level controllers ensure safety and robustness if the planned path is deemed unsafe.

6.4.2 Reactive Controllers Only

In the second run, only the reactive, local controllers were active, and no global plan was provided from the planner. This control strategy exhibited the well-known and expected behavior of getting stuck in the cul-de-sac without any global information (aside from the heading to the goal) to guide the robot (Figure 22(b)). It should be noted though that the safety controller did in fact veto the opening that the planner-only controller tried to push through.

6.4.3 Global Planner with Reactive Controllers

In this scenario, there exists the potential for planning out of the cul-de-sac as well as proper maintenance of safe operation, but since the planner had no way of knowing that the opening was too narrow, it continued to plan through the narrow opening. Meanwhile the reactive controllers vetoed that action. As a result, the robot did not exhibit any improved behavior over the reactive-controller-only situation (Figure 22(c)).

6.4.4 Global Planner with Reactive Controllers and Conflict Detection

Here, the planner once again tried to force the robot through the opening in the cul-de-sac wall. However, the safety-controller vetoed this action as well as encoded this veto through the conflict detection mechanism as an obstacle in the map, and the planner then re-planned its course of action. As seen in Figure 22(d), after a bit of exploring of the cul-de-sac, the robot decided that there was no way forward through the cul-de-sac, and a path was planned out from the area, which enabled the robot

to continue its mission.

6.5 Discussion

In this work, we argue that it is beneficial to introduce a mechanism to detect and correct conflicts between the deliberative and reactive layers of a hybrid control architecture. In this instance, based on the performance of the reactive controllers (and their interaction with the environment), kinematic obstructions are encoded in the global map even though they may not be perceived as obstacles within the planner's model of the robot. Due to the computational burden of global path planning, mapping and planning are often performed in a reduced state space, which means that high degree of freedom kinematics, dynamic constraints, or complex configuration spaces cannot always be handled directly. However, by detecting conflict between the reactive and deliberative layers, they can be incorporated in this lower-dimensional description of the environment. We illustrate this view through an experiment in which a robot is trying to negotiate a cul-de-sac. This experiment shows that the detection and correction of conflict between layers has a beneficial impact on the robot performance.

CHAPTER VII

LEARNING APPLIED TO INTER-LAYER CONFLICT

7.1 Introduction

While the technique of detecting and correcting inter-layer conflicts between deliberative and reactive layers proposed in Chapter 6 proved effective, it is obviously lacking. Foremost, it evaluates instances of inter-layer conflict on an instance-by-instance basis. It does not allow the robot the ability to generalize about the types of situations that cause conflicts between the models of the robot used in the deliberative and reactive layers. If a large number of conflicts potentially exist within the environment, the robot must explore each of them, marking each in its map, and continuing on to the next.

It seems obvious that the robot could do better. As an example, once the robot has experienced conflict between its deliberative and reactive layers, it ought to be able to predict where else in the environment it is likely to also experience similar conflict. This requires that the robot be able to generalize about its experiences, learn concepts that lead to conflict, and modify its plans to avoid these situations accordingly.

The system presented in Chapter 6 detected locations in the terrain map where the deliberative and reactive layers disagreed on whether the robot could navigate the region. In this chapter, we propose a system that detects when the deliberative layer's assumptions about the optimality of its solution are violated. The angle we will exploit is to modify the deliberative layer's dynamical model of the robot. This will aid the deliberative layer in producing plans that are closer to optimal at the system level, with respect to time of travel.

While we argued in Chapter 6 that it was acceptable, and even appropriate, for the deliberative and reactive layers to use different models of the robot's dynamic and kinematic constraints, in this chapter we look to close the performance gap between the models without increasing the computational complexity of the models. Toward this goal, we explore the prospects for learning velocity models based on a supervisory signal provided by the reactive layer acting on plans provided by the deliberative layer. We then exploit these models in the deliberative layer to improve system level performance [40] [39].

7.2 Approach

Our approach to the problem of learning from and planning with respect to conflict between control layers begins with the definition of the experience, described in Chapter 3. Given a representation of an experience, a model must be built relating proposed experiences to similar experiences that the robot has already evaluated. Within a supervised learning paradigm, the robot builds this model based on experiences gathered by executing proposed planning experiences. This can be achieved by allowing the baseline deliberative layer to produce plans that the reactive layer uses as input to navigate the environment. The supervisory signal is provided by the reactive layer's interpretation of each experience. This supervisory signal should reflect the concept to be learned. i.e., if the concept to be learned is the average speed of the robot as it executes the proposed planning experience, the supervisory signal should be designed to evaluate the average speed of the robot.

Once a model of the reactive layer's interpretation of planning experiences has been built, the planner can exploit this model by using it to evaluate proposed experiences in the planning stage. Because the planner works to propose a globally optimal navigation path through the environment, using an improved model of how the proposed path will be interpreted will work to improve the performance of the

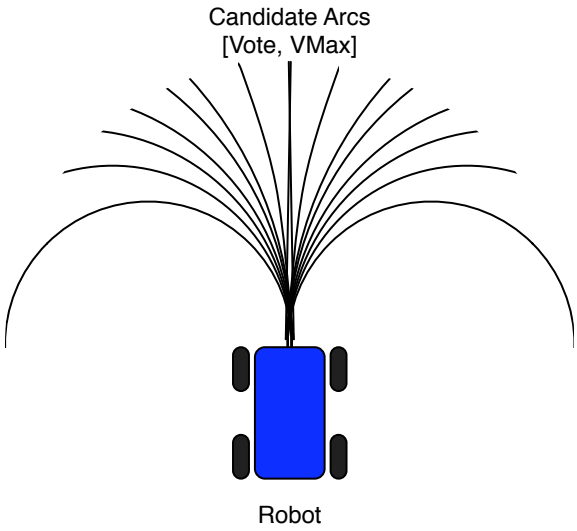


Figure 23: The reactive layer reasons over constant curvature arcs. Associated with each arc is a maximum allowable velocity along that arc. The reactive layer uses a behavior-based approach to choose a curvature and velocity. Each behavior expresses its preference for each candidate arc by allocating its share of votes either for or against each arc. Each behavior also sets a maximum allowable velocity along each arc. The arbitrator adds up the votes for each arc from each behavior, selecting for execution the arc with the most votes. The arbitrator also selects a velocity for execution by taking the minimum of the maximum allowable velocities set for the chosen arc.

overall system.

7.3 Implementation

This system was developed on the simulation implementation of the Sting platform described in Chapter 5. The configuration of the reactive controllers and the deliberative layer is described below.

7.3.1 Reactive Layer

The reactive layer is implemented in a behavior-based voting design, explained in detail in Chapter 5 and [56]. In this design a number of independent behaviors evaluate candidate actions over a short temporal scale, each behavior representing a specific interest pertaining to the robot's objective.

Beginning with dynamical modeling of the system, the robot is modeled as moving through the world in in the x-y plane, and having an orientation within that plane,

$$x = \left\{ \begin{array}{c} x \\ y \\ \theta \end{array} \right\}$$

The robot's dynamics are modeled in this space using an Ackermann steering model, where a steering angle translates into a curvature, κ through the x-y plane,

$$f(x) = \left\{ \begin{array}{l} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ v \cdot \kappa \end{array} \right\}$$

The robot's translational and rotational velocities are modeled as coupled, because the robot uses an Ackermann steering model. The rotational velocity is a function of the product of the controlled translational velocity, v, and the controlled steering curvature, κ, which are also the output of the controller,

$$g(x) = \left\{ \begin{array}{c} v \\ \\ \kappa \end{array} \right\}$$

Following from the dynamical model, the behaviors reason over constant curvature arcs, as shown in Figure 23. In operation, each behavior distributes an allocation of "votes" over an array of potential arcs for the robot to navigate along. The behaviors can allocate votes for arcs that work to achieve its interests, or against arcs that are detrimental to its interests. In addition to distributing votes for or against arcs, behaviors assign a maximum allowable velocity, associated with each arc. Behaviors need not necessarily express an interest across both curvature and velocity. A behavior may vote for curvatures and leave the allowable velocities set to the robot's maximum velocity, it may cast no votes for or against curvatures and express its interest across the allowable velocities, or it may express its interest across both dimensions.

To choose a curvature and velocity for the robot to execute, an arbiter sums the votes cast by each behavior for each curvature arc, weighting the votes for each behavior according to a predetermined weighting scheme. It selects for execution the curvature arc with the highest total of votes. It then selects for execution the minimum of the maximum allowable velocities assigned by the respective behaviors to the selected curvature arc. The selected curvature and velocity are sent on to low-level controllers for execution.

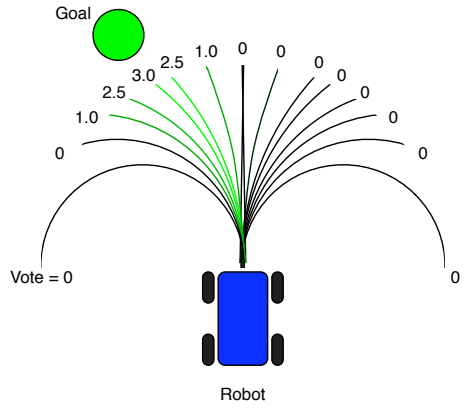
Five behaviors were used in this experiment (shown in Figure 24):

Move to Waypoint - allocates positive votes to arcs according to a linear control law relating the local heading to the waypoint to a commanded curvature. The votes are allocated according to a Gaussian distribution around the output of the linear control law.

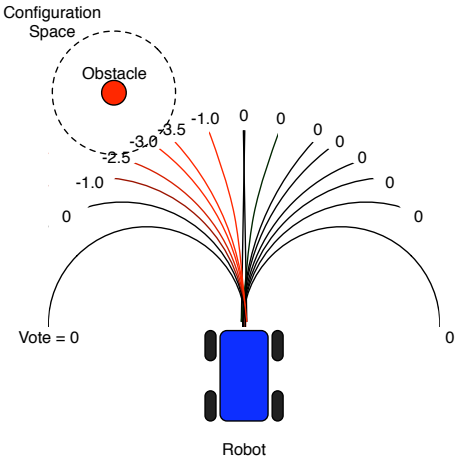
Avoid Obstacles - allocates negative votes to arcs according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. Arcs that do not cross into the configuration space of the obstacle are not voted against.

Maintain Headway - sets maximum allowable velocities for each arc according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. If the arc does not cross into the configuration space of the obstacle, the robot's maximum speed is assigned. If the arc crosses into the configuration space of the obstacle within a parameterized safety distance, the maximum allowable velocity is zero.

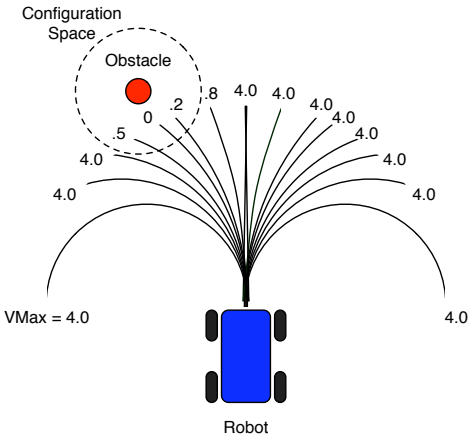
Slow for Congested Areas - sets maximum allowable velocities for each arc according to the distance along the arc that the arc crosses into an intentionally large configuration space around a detected obstacle. If the arc does not cross into the the configuration space of the obstacle, the robot's maximum speed is



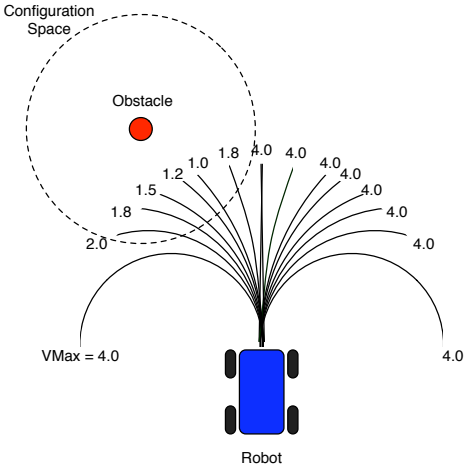
(a) The *move to waypoint* behavior.



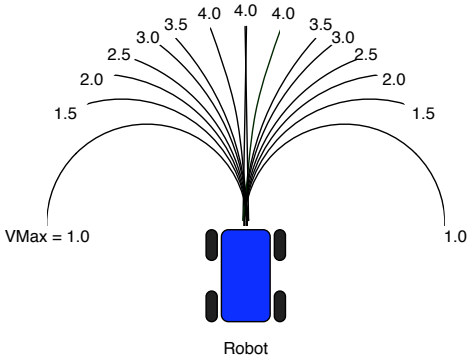
(b) The *avoid obstacles* behavior.



(c) The *maintain headway* behavior.



(d) The *slow for congested areas* behavior.



(e) The *slow for turns* behavior.

Figure 24: The five behaviors used together in in the reactive layer.

assigned. A velocity of zero is never assigned. That responsibility is left to the maintain headway behavior.

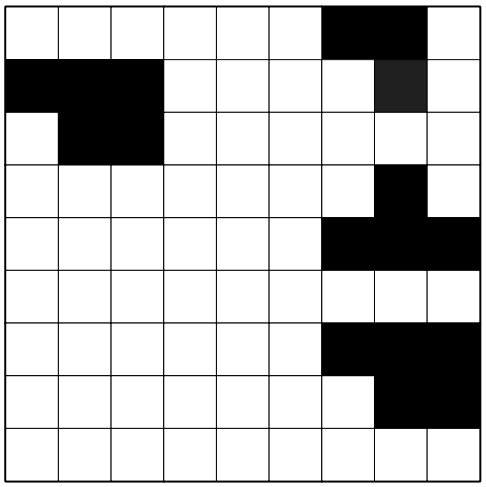
Slow for Turns - sets a maximum allowable velocity for each arc according to a parameterized maximum allowable rotational velocity. if the calculated maximum allowable velocity is larger than the robot's top speed, the robot's top speed is assigned.

7.3.2 Deliberative Layer

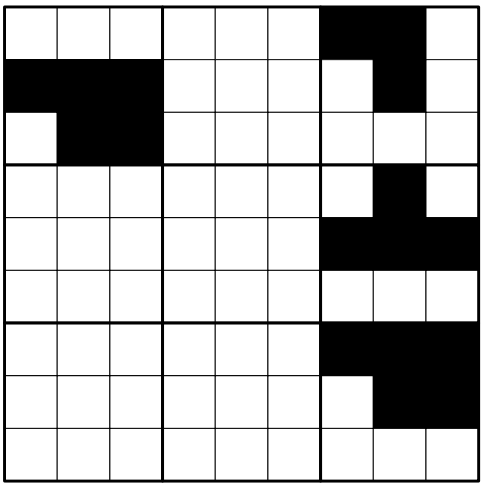
To demonstrate our approach, we implement the deliberative layer as a global path planner over a relatively high-resolution occupancy grid, as shown in Figure 25(a). In this case, the world, W, is implemented as a 120×120 meter area in the x-y plane. The map, M is implemented as an occupancy grid with square grid cells at a .4 meter resolution. As sensory information is accumulated in the local frame it is integrated into the global map based on the robot's current global localization estimate. Detected obstacles are placed into grid cells based on their discretized global position. Each grid cell can be marked as either occupied or unoccupied. Obstacles associated with unoccupied cells cause the cell to be marked as occupied. Obstacles associated with occupied cells have no effect on the cell; the cell remains marked occupied.

The grid cells are then grouped into regions, $r \in R$ as depicted in Figure 25(b). The set of regions, R, is implemented at a 2.0 meter resolution. The size of the regions is chosen to correspond to the two degree of freedom configuration space of the robot, as referred to in Chapter 6. This allows the planner to plan over these regions, the configuration space already accounted for in the discretization of the map. A count of occupied grid cells within each region is kept. This count is used by the planning algorithm in evaluating the traversability of each region.

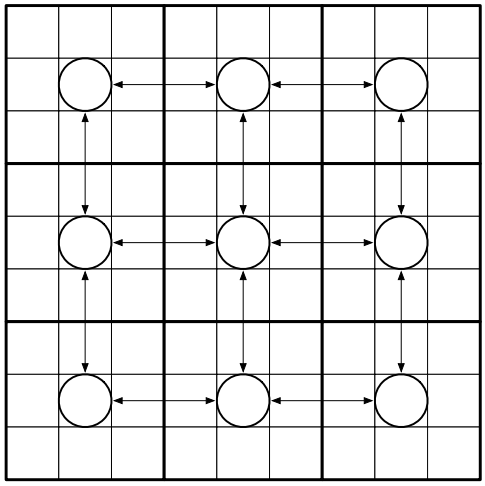
To represent the connectivity between the regions, a graph is overlaid on the map,



(a) The implemented global planner works over a high-resolution occupancy grid. Black grid cells represent occupied cells (obstacles). White grid cells represents unoccupied cells (free space).



(b) The high-resolution grid cells are grouped into larger regions. The size of these regions correspond to the two degree of freedom configuration space of the robot.



(c) The connectivity between regions is represented by an overlaid graph. Here, a four-connected graph is shown overlaid on the grid. The cost of traversing an edge is proportional to the expected time to travel between its two vertices. The baseline planner use a binary model of the robot's velocity. That is, the robot travels at top speed unless its path is blocked by two or more occupied grid cells.

Figure 25: The deliberative layer works over a map that takes advantage of both grid and graph representations.

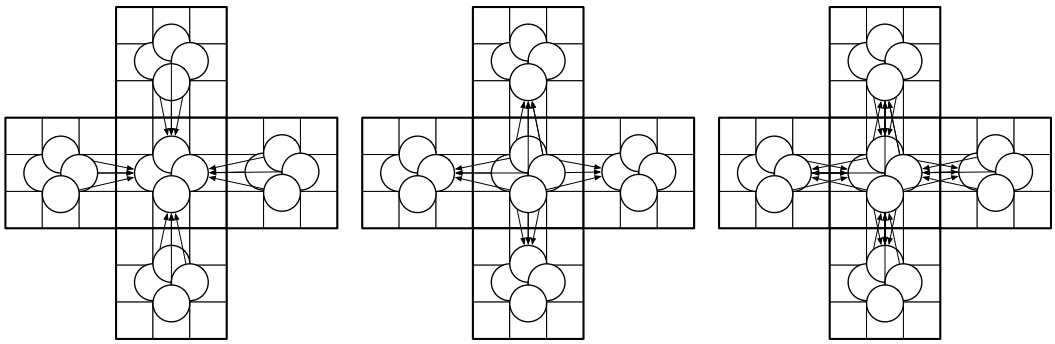


Figure 26: To represent an approximation of the robot's heading in the two degree of freedom map, four vertices are "stacked" on each region. Each vertex is receives edges originating from vertices in one of the four possible directions. (a) shows just the edges coming into the center region. (b) shows just the edges originating from the center region. (c) shows the super set of (a) and (b), all edges originating from or ending in the center region.

as shown in Figure 25(c). One graph vertex is associated with each region. Edges are added between contiguous vertices. Figure 25(c) depicts a four-connected graph based on the region structure of the occupancy grid. This graph structure is a suitable data structure for many planning algorithms. In this implementation, an instance of the D*-Lite [31] algorithm is employed to choose the mapping from regions to controllers described in Equation 9,

$$b = \arg\min_{b \in B} \sum_{I \in \tau} c(r_I^o, M, b(r_I^o)) \mid Acc(\chi) = True.$$

The set of controllers in each region, G_{r^o}, is implemented as a set of behavioral assemblages, using the behaviors described above, with the *move-to-waypoint* behavior parameterized with each neighboring region's center, respectively. The transition model for each controller, $d(r^o, M, g_{r^o})$ returns the neighboring region with which *move-to-waypoint* behavior is parameterized.

The cost of traversing each edge is proportional to the expected time to move between its source vertex and destination vertex. The time to move between vertices is modeled as the distance between the vertices, d divided by the expected average

velocity of the robot over that distance, v. The baseline planner uses a binary model of the robot's velocity. If the count of occupied grid cells within the regions associated with either vertex is larger than 1, the expected velocity is zero (i.e. the edge is not traversable and is assigned an infinite cost). Otherwise, the expected velocity is the robot's top speed, v_{max}.

$$c_0(r^o, M, g) = \begin{cases} \frac{d}{v_{max}} & if \sum_{x \in r^o} M_t(x) > 1 \\ \infty & else \end{cases} \tag{15}$$

where $M_t(x)$ is the value in the occupancy map at location x, at time t.

7.3.3 Learning

To make the learning component of this approach tractable, it is important to create a compact, yet meaningful representation of the robot's experiences executing proposed planning segments. Within the format of the of the experience structure proposed in Chapter 3, we propose the following implementation of the experience:

command - the next waypoint

end criteria - transitioning into a neighboring region

context - the local map around the current waypoint (a section of occupancy grid)

measured outcome - the measured average velocity toward the next waypoint

To achieve this, it is necessary to be able to orient the graph into the robot's local frame. That is, rather than encoding the segments of the plan as "move north, move east", it is more general to encode the robot's experiences as "move forward, move right". A more general encoding of experiences makes learning over these experiences easier, as it reduces the dimensionality of the problem by exploiting symmetry in the data.

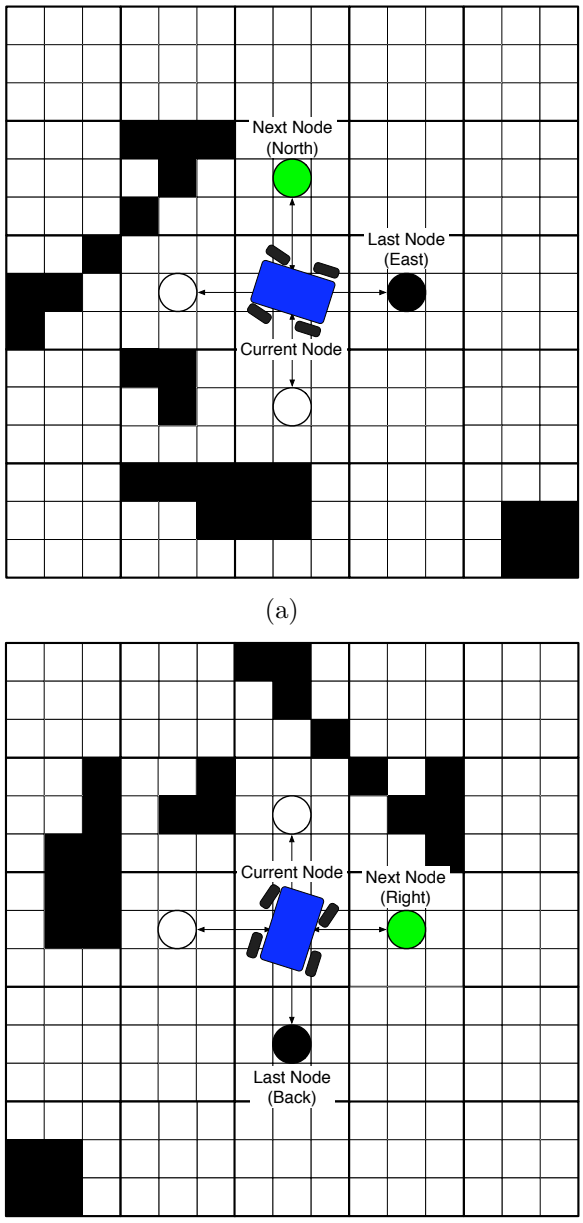


Figure 27: Experiences are encoded for learning by orienting the local map based on the waypoint achieved previous to the waypoint associated with the robot's current position in the map. This is alway derivable, because of the redundant graph vertices depicted in Figure 26. The unrotated experience is shown in (a). The last transition was from East to West. The next commanded transition is from South to North. This experience has been rotated in (b), to represent a right turn in the local frame.

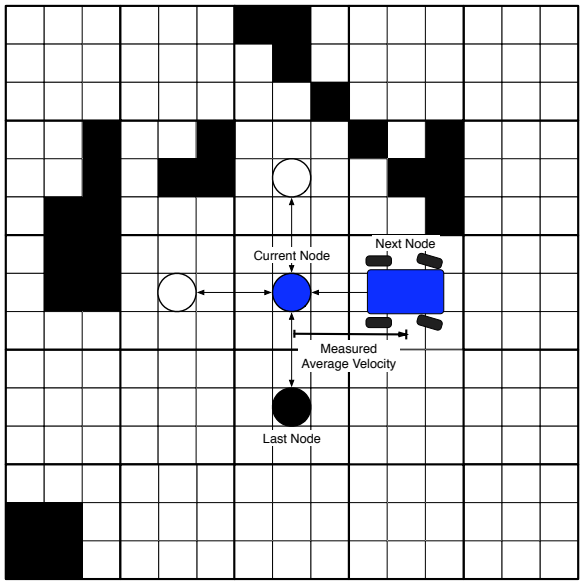


Figure 28: When the robot achieves a new waypoint, the planning experience is complete. The robot's state is recorded, including the oriented local map and the direction of the commanded waypoint (forward, backward, left or right). In this example, the robot was commanded to move from the blue vertex in the center of the diagram to the next vertex at the right of the diagram. The average velocity of the robot over that experience is recorded as the supervisory signal.

Creating this general representation requires at least a coarse representation of the robot's rotational orientation. In this implementation, this is achieved by "stacking" multiple vertices, one for each direction of connectivity, in the same location. Each stacked vertex is associated with the same region, and thus, the same global location. Figure 26 depicts this design. In this four-connected graph implementation, each of these four stacked cells serves as the destination vertex for all directional edges coming from one direction of connectivity, as shown in Figure 26(a). i.e., the first stacked vertex receives edges coming from vertices to the West. The second stacked vertex receives edges coming from vertices to the North, and so on. Each stacked vertex also serves as the source vertex for directed edges in each direction of connectivity, as shown in Figure 26(b).

Using this representation, it is possible to project the robot's current state into a local representation by rotating with respect to the last edge traversed. The direction of the last edge traversed can always be recovered by checking from which direction the robot's current vertex accepts incoming edges. Then, the next commanded waypoint can be characterized as a "left-turn" or "right-turn" based on its relative direction in the local frame. For example, as shown in Figure 27, if the last edge traversed was from west to east (Figure 27(a)), when the map is put into a local frame (Figure 27(b)), the next waypoint, which is to the north, will be represented as a right turn.

With the planning experience placed into the robot's local frame, the experience can be recorded for the learning stage. Figure 28 is a graphical representation of the robot's learning experience. The vertex in which the robot started the experience is shown at the center of the diagram, colored blue. The waypoint commanded by the global planner is shown above that vertex, colored green. This commanded waypoint would be encoded in the planning experience as a "go forward" command. This is because it represents a forward movement assuming that the robot last traversed the edge originating from the vertex at the bottom of the image, colored black. As

explained above, the experience is orientated with respect to this last completed traversal. The underlying map grid cells are also placed into this local frame, and included in the planning experience state space.

The planning experience's supervisory signals are provided by the reactive layer's interpretation of the commanded plan. For example, in Figure 28, the reactive layer's interpretation of the plan in conjunction with the local environment has brought the robot to the commanded "right-turn" waypoint. The recorded planning experience will make a record of the success of the experience, the local section of the occupancy map, and the average speed of the robot during this experience.

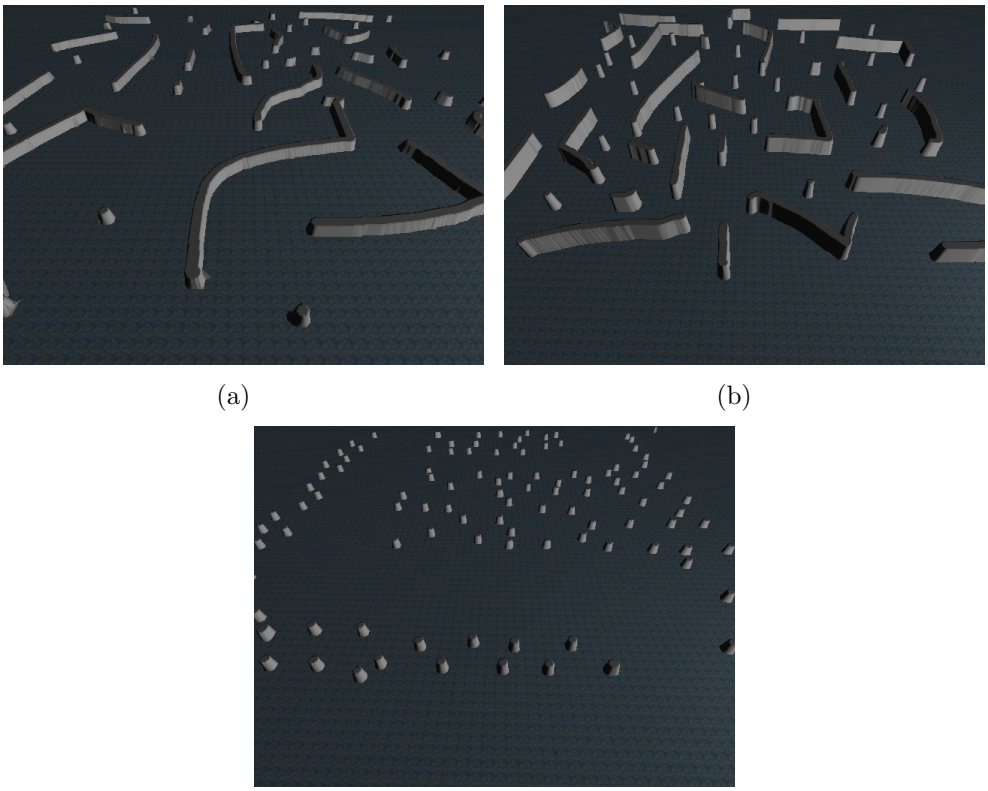
Once a sufficient number of planning experiences have been recorded, the experiences are used as data for a supervised learning algorithm. The supervised learner uses the experiences to extrapolate expected results from new proposed experiences, deriving a new cost model, as in Equation 11,

$$c_t = \arg\min_{c \in C} \; E[\; |h_c(x, r^o) - c(r^o, M, g)| \;] \mid \{h_c(x, r^o), M, g\}^t$$

The learned model of expected velocity is used by the global planner to plan subsequent navigation paths. The planner uses the model to evaluate the expected cost of traversing an edge, in terms of expected time to traverse the edge. To evaluate the cost of an edge, the edge is encoded in terms of a planning experience, i.e. in terms of being a "left-turn" or "right-turn", etc., along with the oriented local map. The learned velocity model returns an expected velocity over the edge. The time-based cost is derived by dividing the distance between the source vertex and the destination vertex by the expected velocity. The planning algorithm plans over these costs to find the best plan, given c_t, as in Equation 12,

$$b_t = \arg\min_{b \in B} \sum_{I \in \tau} c_t(r_I^o, M, b(r_I^o)) \mid Acc(x) = True, \; \{h_c(x, r^o)\}^t.$$

The learned cost model, c_t may be learned by batch processing (i.e. learn the model only once a large number of learning experiences have been collected) and left



(a) (b)

(c)

Figure 29: The three simulation environments used in testing. (a) is the training environment. (b) is the quantitative testing environment. (c) is the qualitative testing environment.

constant afterwards, or the model may be learned incrementally (i.e. continue learning as soon as a new experience is recorded). While not all learning algorithms lend themselves to incremental updates (e.g. artificial neural networks), other algorithms handle incremental updates easily (e.g. k-nearest neighbors). To demonstrate the value of learning in this context, both strategies will be explored in the experiments presented in this chapter.

7.4 Experiments

To demonstrate the value of the system presented in this chapter, a series of experiments were set up, each demonstrating different aspects of the system. The first experiment quantitatively compared the performance of the experimental system to

a baseline system. The second experiment quantitatively compared the performance of the experimental system using different numbers of training examples. The third experiment qualitatively demonstrated the performance of the experimental system versus a baseline system. The final experiment compared the performance of the experimental system using different cost functions.

As part of these experiments, three complex environments were designed within the simulation environment, the Player/Stage/Gazebo system [24]. The environments used are shown in Figure 29. Figure 29(a) shows the environment that was used for gathering training data. Figure 29(b) shows the environment that was used for quantitative comparative testing. Figure 29(c) shows the third environment, used to demonstrate the qualitative behavior of the experimental system.

Training experiences were gathered in the training environment by tasking the robot to achieve a series of randomly generated goals throughout the environment, using the baseline global planner and the above described reactive layer. Every time the robot achieved a waypoint a learning experience was recorded, including the local map, the robot's average velocity, and the commanded waypoint.

7.4.1 Experiment 1: Experimental System vs. Baseline

7.4.1.1 Experimental Setup

In the first experiment, the performance of the experimental system was compared to a baseline system. In this experiment, time of execution was used as the planning cost function in both the experimental and baseline systems. The baseline system used the cost model described in Equation 15. The experimental system used a learned cost model built from 5000 training examples.

Several different supervised learning algorithms were evaluated for use by the experimental system. The Weka machine learning environment [19] provides a library of community-supported implementations of well known algorithms. For learning prediction of the robot's velocity, we evaluated the k-nearest neighbor regression

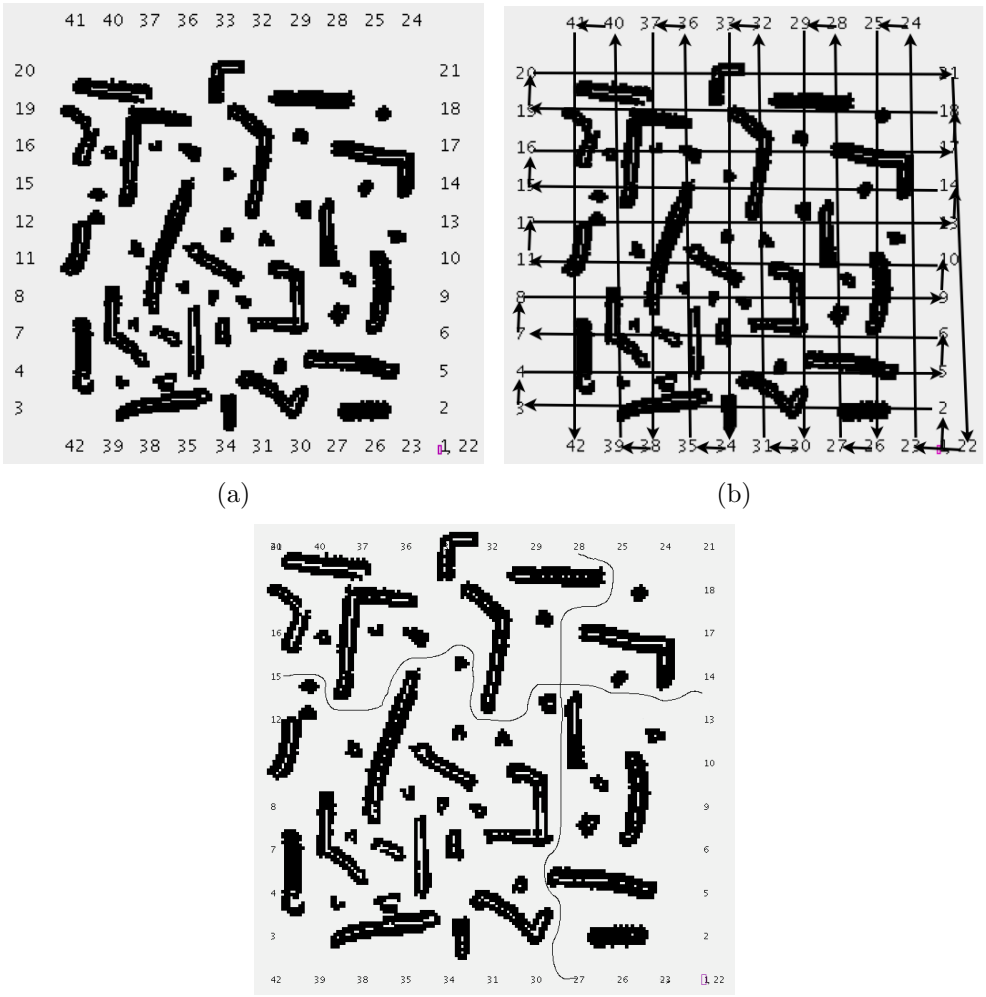


Figure 30: The sequence of goals used in the quantitative tests. (a) shows the position of the goals around the perimeter of the obstacle course. (b) shows the sequential straight-line path between the goals. (c) shows two sample trajectories between goals 14 and 15, and 27 and 28. These sub-trajectories are compared between tests to produce a paired T-test (a statistical measurement of confidence).

algorithm for several values of k, a multi-layer perceptron network, linear regression, support vector machine regression, and the baseline strategy of always assuming the robot travels at its maximum speed.

Table 1: Results of a 10-fold cross validation test on several learners predicting the robot's velocity given a proposed planning transition. The k-Nearest Neighbor algorithm with the k-value set to 5 produced the best results, nearly cutting the average relative error in half, compared to the baseline approach.

	Baseline	KNN $k = 50$	KNN $k = 15$	KNN $k = 5$	KNN $k = 2$	ANN	SVM Reg.	Lin. Reg.
Average Relative Error	106%	80%	72%	64%	61%	91%	77%	86%
Correlation	-.04	.57	.62	.65	.63	.55	.47	.50

Models were built from the training data, using each algorithm. Cross-validation tests on the training data were performed to evaluate the effectiveness of each algorithm. Table 1 displays the cross validation results for each algorithm on the velocity data. The k-nearest neighbor algorithm performed the best of the algorithms tested. While the instance with $k = 2$ produced the lowest average relative error, the instance with $k = 5$ was chosen for use because it produced the highest correlation. Because the planning algorithm will use the model to compare possible routes, it is more important that the model produce highly correlated predictions than that the model produce a low absolute error.

The learned model was then incorporated into the cost model of the experimental system's global planner. Using the k-nearest neighbor algorithm, the learned cost model, c_t is defined as,

$$c_t(r_t^o, M_t, b(r_t^o)) = \frac{\sum_{i=1}^{k} w_i \cdot v_i}{\sum_{i=1}^{k} w_i} \tag{16}$$

where $\{v\}_1^k$ is the set of retrieved k-nearest neighbor values, in this case retrieving the values of the 5 nearest training instances, to the current context, $(M_t, b(r_t^o))$. w_v

is the distance weight given to each neighbor, defined as,

$$w_i = \frac{1}{dist(M_{v_i}, M_t)}$$

and,

$$dist(M_{v_i}, M_t) = \sqrt{\sum_{r^o \in M_t, M_{v_i}} \sum_{x \in r^o} (M_{v_i}(x) - M_t(x))^2}.$$

The baseline system was compared to the new experimental system. Each system was tasked with achieving a sequence of goals in the test environment, as shown in Figure 30(a). This sequence of goals totaled a piecewise straight-line distance of over 1500 simulated meters (Figure 30(b)). Figure 30(c) displays two sub-trajectories between goals 14 and 15, and 27 and 28. These sub-trajectories between respective goals were compared between tests to produce a paired T-test to measure the statistical significance of the measured results. To demonstrate how the performance of the planner improved with experience, tests were then run with increasing numbers of examples used in the learned k-nearest neighbor model. Results were compiled comparing the average time to complete each goal between different systems.

7.4.1.2 Results

Table 2: Results of timed test of baseline and experimental systems. The experimental system improved performance over the baseline by 17%, with a statistically significant p-value of .02.

	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	7745 sec	5124 m	N/A	N/A
Experimental System (KNN $k = 5$)	6434 sec	5577 m	17%	.02

Table 2 displays the results of using the learned models in the global planner on the time to complete the sequence of goals shown in Figure 30(a). Using the learned velocity model reduced the time to complete the overall mission by 17% over the baseline system. The calculated p-value of .02 indicates that the result is statistically significant. The fact that the velocity model does show improvement indicates that using learning to improve the deliberative layer's model of how its plans are interpreted by the reactive layer can improve the robot's overall system performance.

7.4.2 Experiment 2: Performance vs. Number of Training Examples

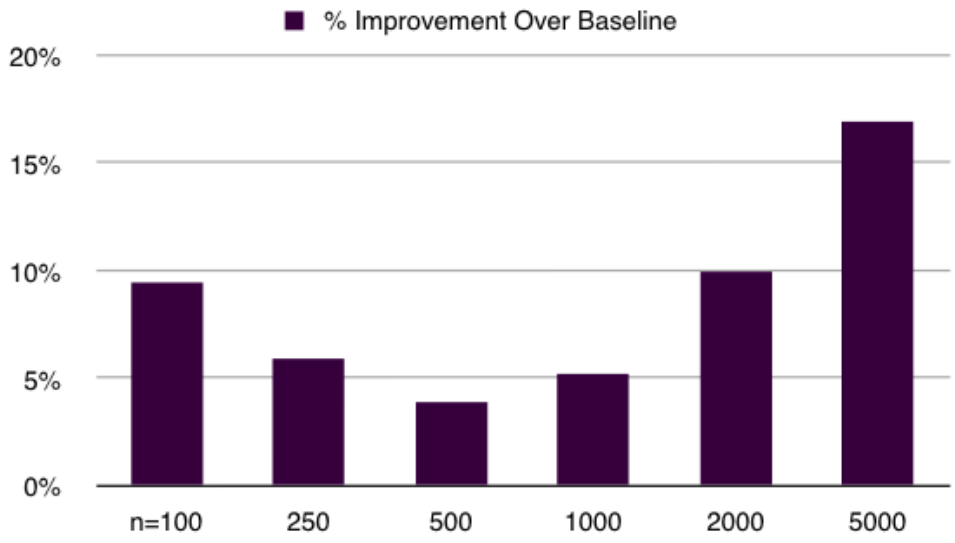
7.4.2.1 Experimental Setup

In the second experiment, the effect of varying the number of training examples used to build the experimental system's cost model was explored. A series of experimental runs were performed. In each run, the experimental system used in Experiment 1 was configured with a cost model built from a different number of training examples. Models built from 100, 250, 500, 1000, 2500 and 5000 training examples were tested. The training examples used for training each model were independently sampled from the full set of training examples used in Experiment 1.

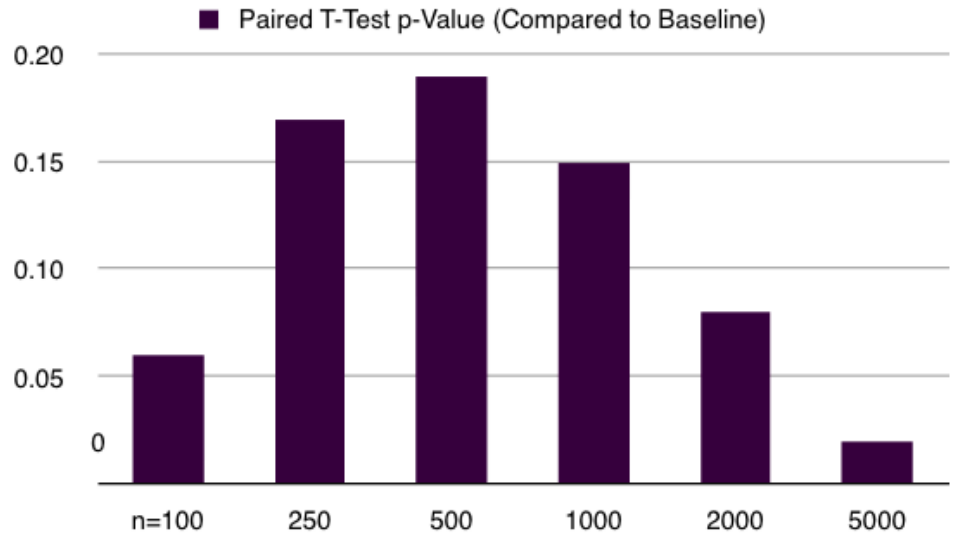
Cross-validation tests on the training data were performed for each model. Table 3 displays the cross validation results for each instance tested. A strong correlation can be seen between the number of training examples and both the average relative error and the correlation between the model and the training data.

7.4.2.2 Results

Each instance of the experimental system was run over the same test course used in Experiment 1. The total execution time for each run was recorded and compared to the performance of the baseline system. Demonstrating how adding experiences affects the performance of the system, Figure 31 shows the trend of performance



(a)



(b)

Figure 31: System performance improvement over the baseline planner, as a function of the number of examples used in training a k-nearest neighbor regression model. (a) shows the improvement over the baseline system, while (b) shows the paired t-test p-value for each instantiation.

Table 3: Results of a 10-fold cross validation test on several k-nearest neighbor instantiations, using varying numbers of examples to train. The value of k was set to $k = 5$ throughout the tests.

	$n = 100$	$n = 250$	$n = 500$	$n = 1000$	$n = 2500$	$n = 5000$
Average Relative Error	112%	95%	87%	74%	71%	64%
Correlation	.33	.43	.49	.54	.56	.65

improvement over the baseline planner. The experimental planner gets an initial boost of improvement with just a few examples, and continues to trend upward as the number of examples is increased. Because k-nearest neighbors is an incremental algorithm, a planner could be implemented to learn online, adding experiences to its model as they are executed.

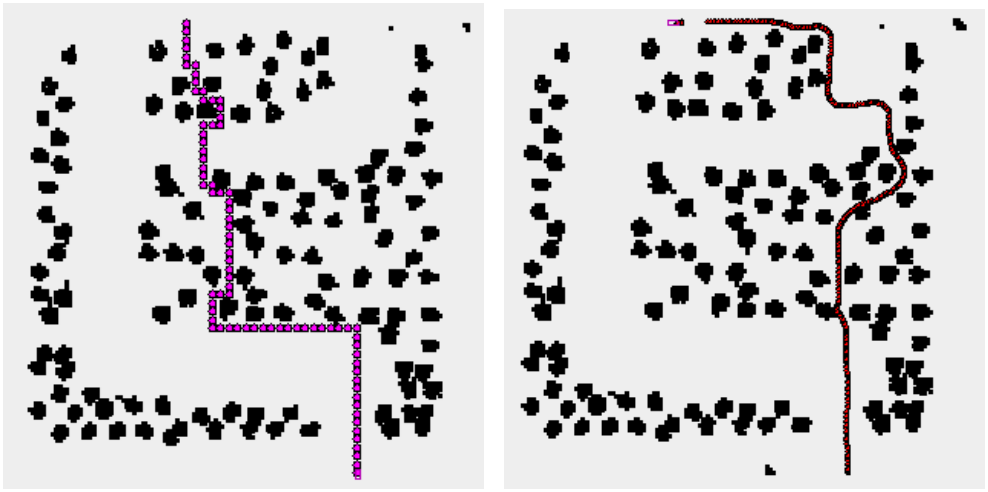
7.4.3 Experiment 3: Qualitative Performance

7.4.3.1 Experimental Setup

In addition to quantitative experiments, a qualitative experiment was performed to demonstrate, in an intuitive way, the effect learning had on the overall system performance. An environment was constructed to resemble an open path through a wooded area, shown in Figure 29(c). The wooded area is sparse enough that the robot is capable of finding a path between the trees, but would travel that path slowly due to its tendency to drive slowly in tight spaces and slow down for the frequent required turns. The robot was tasked with navigating to a goal whose straight-line path would take the robot through the woods. The plans and resulting paths created by the baseline system and the experimental system were compared qualitatively and quantitatively.

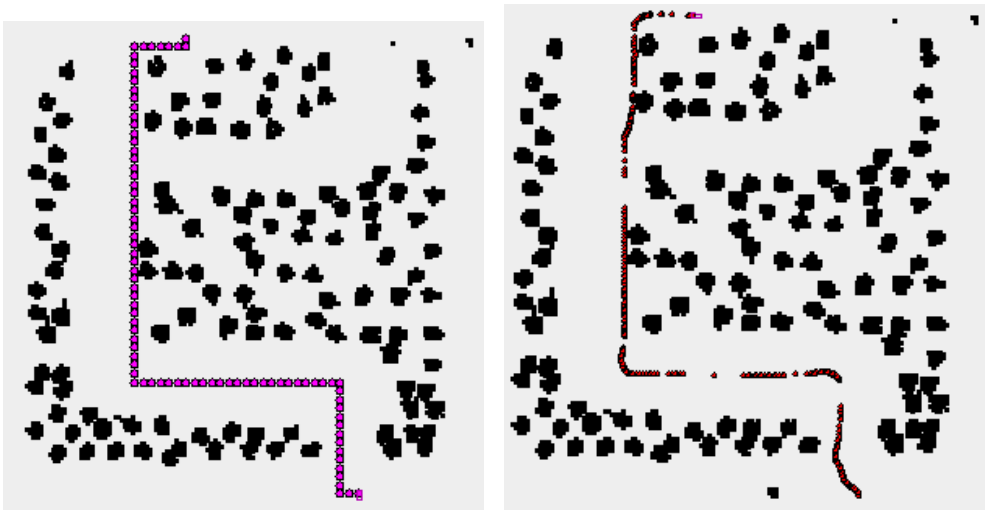
7.4.3.2 Results

Figures 32 and 33 show the results of the qualitative tests in the "path through the woods" environment. Figure 32(a) shows the baseline planner's planned route



(a) (b)

Figure 32: In (a), the planned path from the robot to the goal through the "path in the woods" environment using the baseline planner. The robot is at the bottom of the image. The goal is at the top of the image. The planned path is shown by pink waypoints. Obstacles in the map are shown in black. In (b), the actual path taken by the system (trajectory in red).



(a) (b)

Figure 33: In (a), the planned path from the robot to the goal through the "path in the woods" environment, using the experimental planner. The resulting plan is longer given a constant velocity model of the robot, but when used as input to the reactive layer, as shown in (b), reduces mission time by 25% over the plan shown in Figure 32(a).

through the environment. Note how the planned route snakes through the dense obstacle field on its way to the goal. Figure 32(b) shows the route actually taken by the robot following the planner's output. Note that it departs from the planned route early in the mission. The planner continues to suggest updated routes based on the robot's position.

Figure 33(a) shows the planned route provided by the experimental system. Note that it prefers a slightly longer (by distance) route that follows the wide path. Figure 33(b) shows the route actually taken by the robot following this plan. In this trial, the robot completes the mission in 25% less time than the baseline planner. This demonstrates a clear qualitative and quantitative improvement in system-level performance in a plausible environment.

7.4.4 Experiment 4: Performance vs. Cost Function

7.4.4.1 Experimental Setup

Noting that the first three experiments all made use of the same cost functions, a final experiment was performed to demonstrate the experimental system's ability to learn different planning cost functions and to demonstrate qualitatively different behavior by systems built by learning the different cost functions. In this experiment, experimental systems trained on two different fuel models were compared to each other and a shortest-path baseline system.

The cost functions used in this experiment used a simple model of power consumption, based on the development of an automobile power model in [58]. Since the power for an automobile comes from its fuel, a power consumption model can be seen as an idealized fuel consumption model (given that inefficiencies in fuel combustion are ignored). This power model takes into account three terms. First, is a constant term, representing the overhead for keeping an engine running (e.g. the power required to idle the engine),

$$P_{overhead} = c \tag{17}$$

where c is a constant power drain on the vehicle. The second term is a representation of the power required to accelerate the vehicle,

$$P_{accel} = m \cdot \ddot{x}^{+}(t) \cdot \dot{x}(t) \tag{18}$$

where m is the mass of the vehicle, and $\ddot{x}^{+}$ is the positive component of the acceleration of the vehicle. (i.e., in this model, deceleration is considered "free", just as braking has little effect on a car's fuel consumption). The final term represents the power required to overcome aerodynamic drag on the vehicle,

$$P_{drag} = \frac{1}{2} \cdot C \cdot A \cdot \rho \cdot \dot{x}(t)^3 \tag{19}$$

where C is the vehicle's coefficient of drag, A is the area of the vehicle's footprint in the translational direction, and ρ is the density of the air. To keep this model simple, rolling friction and internal inefficiencies are ignored. Given these assumptions, the model of fuel usage over the traversal of a region can be written as

$$P = P_{overhead} + P_{accel} + P_{drag} = c + m \cdot \ddot{x}^{+}(t) \cdot \dot{x}(t) + \frac{1}{2} \cdot C \cdot A \cdot \rho \cdot \dot{x}(t)^3 \tag{20}$$

We can simplify the notation somewhat, by collapsing the constant factors into "weights" on each dynamical term,

$$P = w_1 + w_2 \cdot \ddot{x}^{+}(t) \cdot \dot{x}(t) + w_3 \cdot \dot{x}(t)^3 \tag{21}$$

where w_1 has the units of watts, w_2 has the units of kilograms, and w_3 has the units of kilograms per meter (area times density times a unit-less coefficient of drag).

Using this model of power (and, by proxy, fuel usage), three cost measurement functions were created by choosing the constants w_1, w_2 and w_3 to emphasize one of the terms of the fuel usage model, respectively. In cost measurement function 1, h_{c1}, w_1 was set to 100, w_2 was set to 1 and w_3 was set to 1. This function emphasizes the overhead required to keep the vehicle running. In cost measurement function 2, h_{c2}, w_1 was set to 1, w_2 was set to 100 and w_3 was set to 1. This function emphasizes the

power required by accelerating. Finally, in cost measurement function 3, h_{c3}, w_1 was set to 1, w_2 was set to 1 and w_3 was set to 100. Thus, the cost to traverse a region, r^o according to each cost function is,

$$\int_0^T h_{c1}(x, r^o) = \int_0^T 100 + \int_0^T 1 \cdot \ddot{x}^+(t) \cdot \dot{x}(t) + \int_0^T 1 \cdot \dot{x}(t)^3(t) \tag{22}$$

$$\int_0^T h_{c2}(x, r^o) = \int_0^T 1 + \int_0^T 100 \cdot \ddot{x}^+(t) \cdot \dot{x}(t) + \int_0^T 1 \cdot \dot{x}(t)^3(t) \tag{23}$$

$$\int_0^T h_{c3}(x, r^o) = \int_0^T 1 + \int_0^T 1 \cdot \ddot{x}^+(t) \cdot \dot{x}(t) + \int_0^T 100 \cdot \dot{x}(t)^3(t) \tag{24}$$

where T is the amount of time to traverse region r^o.

Models were built from training data using several algorithms from the Weka machine learning environment, including k-nearest neighbor regression, a multi-layer perceptron network, and linear regression, and a baseline strategy of always predicting the robot's average fuel consumption. Tables 4, 5 and 6 summarize the performance of these algorithms on a 10-fold cross validation test on the training data. In all cases, instances of the k-nearest neighbor algorithm provided the best correlation with the training data, and were used in the experiments.

Table 4: Results of a 10-fold cross validation test on several learners predicting the cost function h_{c1}, given a proposed planning transition. The k-Nearest Neighbor algorithm with the k-value set to 5 produced the best results, nearly cutting the average relative error in half, compared to the baseline approach, and producing a very high correlation coefficient.

	Baseline	KNN $k = 50$	KNN $k = 15$	KNN $k = 5$	KNN $k = 2$	ANN	Lin. Reg.
Average Relative Error	100%	63%	59%	57%	57%	85%	81%
Correlation	-.04	.62	.65	.66	.65	.50	.49

These learned models were then incorporated into the cost model of the experimental system's global planner. Each system compared to a baseline system using

Table 5: Results of a 10-fold cross validation test on several learners predicting the cost function h_{c2}, given a proposed planning transition. The k-Nearest Neighbor algorithm with the k-value set to 5 produced the best results, producing the highest correlation coefficient.

	Baseline	KNN $k = 50$	KNN $k = 15$	KNN $k = 5$	KNN $k = 2$	ANN	Lin. Reg.
Average Relative Error	69%	79%	74%	73%	74%	111%	92%
Correlation	-.02	.56	.60	.60	.58	.38	.44

Table 6: Results of a 10-fold cross validation test on several learners predicting the cost function h_{c3}, given a proposed planning transition. The k-Nearest Neighbor algorithm with the k-value set to 15 produced the highest correlation coefficient.

	Baseline	KNN $k = 50$	KNN $k = 15$	KNN $k = 5$	KNN $k = 2$	ANN	Lin. Reg.
Average Relative Error	100%	70%	62%	58%	73%	96%	82%
Correlation	-.02	.61	.64	.64	.62	.47	.51

the baseline model of fuel usage. Each system was evaluated over the same goals and environment used in Experiment 1. Results were compiled comparing the fuel usage for each system.

7.4.4.2 Results

Table 7 compares the results of running the above described test with the baseline system and the experimental system using cost measurement function 1 (h_{c1}). Using the learned cost model, the experimental system was able to reduce the fuel used by 12.7% over the baseline system. The paired T-test p-value of .04 indicates a statistically significant result.

Table 8 compares the results of running the above test with the baseline system

Table 7: Results of the experiment comparing the performance of the baseline system and the experimental system, using cost measurement function 1, h_{c1}. The experimental system improved performance over the baseline by nearly 13%, with a statistically significant p-value of .04.

	Simulated Fuel to Complete Test	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	585,606.86 units	6437 sec	4026 m	N/A	N/A
Experimental System (KNN $k = 5$)	511,158.26 units	5879 sec	3922 m	12.7%	.04

and the experimental system using cost measurement 2 (h_{c2}). In this test, the experimental system was able to reduce the fuel used by 30.5%. The paired T-test p-value of .000002 indicates an extremely significant result.

In Figures 34 and 35, the trajectory of the baseline and experimental systems are compared. Figure 34(a) shows the trajectory of the robot over the sequence of goals used in the quantitative testing. Compare that trajectory to the trajectory created by experimental system using the learned cost model based on fuel model h_{c1}, shown in 34(b). In addition to a convergence of the trajectory along learned low-cost paths, note the general tendency for robot to avoid driving toward or close to obstacles. Also, note the emergence of regions that the robot learns to avoid as high-cost regions. Figures 35(a) and 35(b) demonstrate a similar pattern for the experimental system learned based on the fuel model h_{c2}.

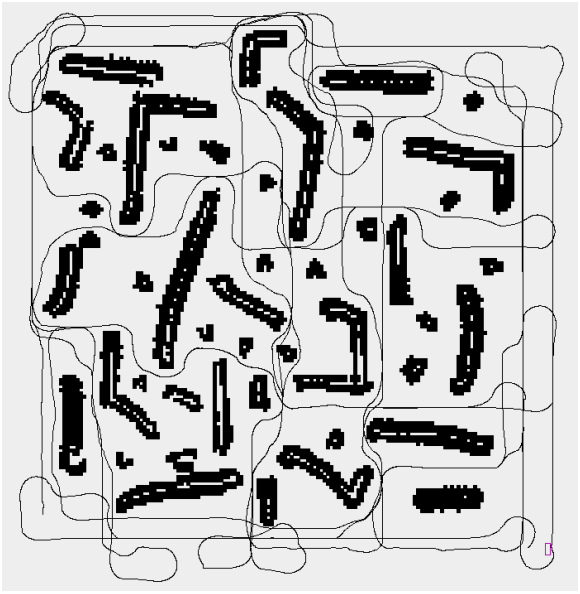
In contrast to the configurations using the cost measurement functions h_{c1} and h_{c2}, the experimental system using h_{c3} performed worse than the baseline system. Table 9 compares the performance of the baseline system and the experimental system. The experimental system finished the test in less time than the baseline system. However, because the cost function was primarily dependent on velocity, the cost

Table 8: Results of the experiment comparing the performance of the baseline system and the experimental system, using cost measurement function 1, h_{c2}. The experimental system improved performance over the baseline by 30%, with a (very) statistically significant p-value of . 000002.

	Simulated Fuel to Complete Test	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	248,187.99 units	6437 sec	4026 m	N/A	N/A
Experimental System (KNN $k = 5$)	172,472.05 units	5657 sec	4015 m	30.5%	.000002

incurred was actually nearly 17% higher. This is not surprising, since the planner in this implementation does not have control over the robot's velocity. The best it can do is guide the robot into obstacle-dense areas, with the expectation that the robot will have to slow down. In Figure 36, the trajectories of the baseline system and the experimental system using h_{c3} are compared. Note that the trajectory does not avoid obstacle-dense areas, as the other experimental systems did.

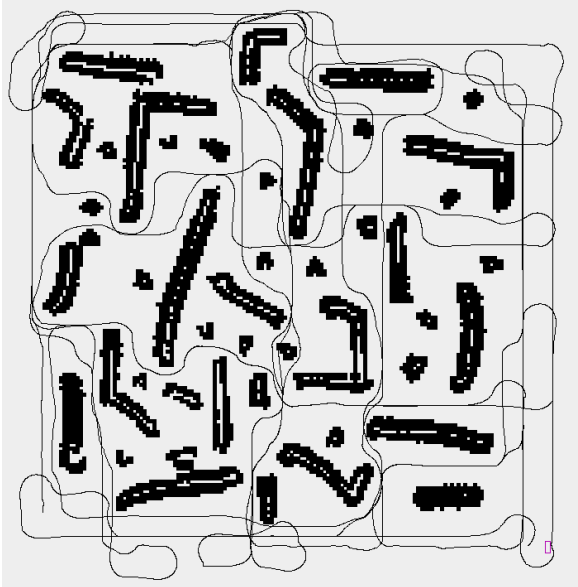
This set of results demonstrates the experimental system's ability to learn a cost functions not only heavily influenced by time, but also acceleration. It also demonstrates some of the limitations of the experimental system. The experimental system was shown to be able to learn cost functions based on a simple fuel model, resulting in a system that produces spatial plans that result in lower fuel consumption when executed. However, the experimental system was unable to learn a cost function heavily influenced by velocity, possibly because the learned planner does not have control of the velocity of the robot. While this experiment does not demonstrate applicability to any general set of cost functions, it does provide evidence that the system is capable of learning a set of cost functions larger than just time of execution.



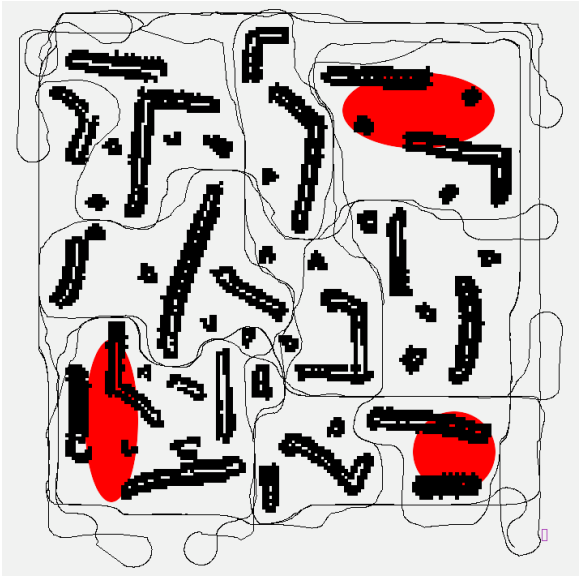
(a) Baseline System

(b) Experimental System

Figure 34: A comparison of the trajectories produced by the baseline system, (a), and the experimental system trained using the fuel model h_{c1}, (b). Qualitatively, one can note that the the experimental system converges to several well-trod, low-cost paths through the environment. It also spends less time driving toward obstacles, and avoids three high-cost areas (highlighted in red) that the baseline system navigates through multiple times.

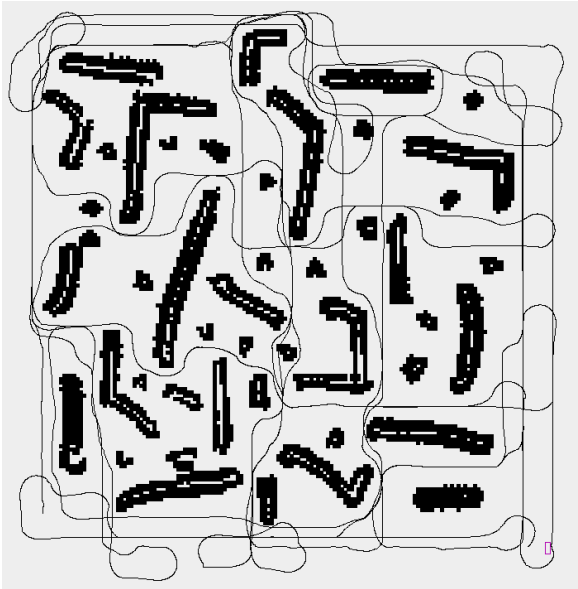


(a) Baseline System

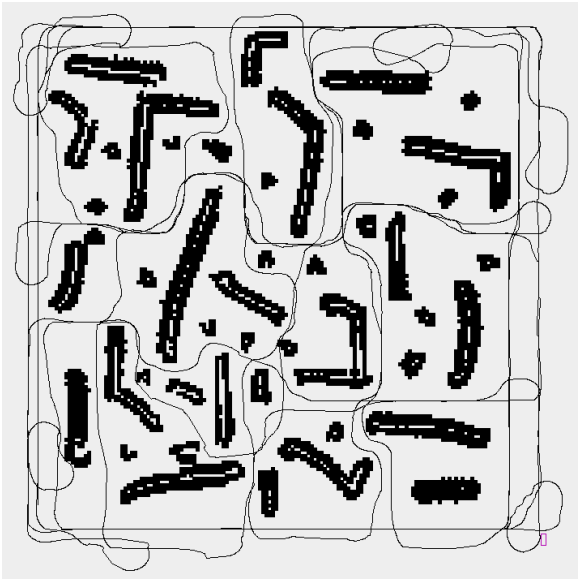


(b) Experimental System

Figure 35: A comparison of the trajectories produced by the baseline system, (a), and the experimental system trained using the fuel model h_{c2}, (b). Like the experimental system trained on fuel model h_{c1}, this experimental system converges to low-cost paths through the environment. Again, it spends less time driving toward obstacles and avoids three high-cost areas (highlighted in red) that the baseline system navigates through multiple times.



(a) Baseline System



(b) Experimental System

Figure 36: A comparison of the trajectories produced by the baseline system, (a), and the experimental system trained using the fuel model h_{c3}, (b). Like the other two cost functions, one can note that the the experimental system converges to several well-trod, through the environment. However, because the cost function is heavily weighted by velocity, and the planner does not have direct control over the robot's velocity, the resulting trajectory does not reduce the overall cost.

Table 9: Results of the experiment comparing the performance of the baseline system and the experimental system, using cost measurement function 1, h_{c1}. The experimental system performed worse than the baseline by 17%. This result, however is not surprising, as the learned planning function does not have control over the primary component of the cost function, velocity.

	Simulated Fuel to Complete Test	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	1,846,857.7 units	6437 sec	4026 m	N/A	N/A
Experimental System (KNN $k = 15$)	1,582,452.9 units	5867 sec	4183 m	-16.7%	.04

7.5 Discussion

In this chapter, we outlined an approach to generalizing upon and correcting inter-layer conflict. We detailed an implementation of this approach in a simulation environment based on a fielded robotic system. Several quantitative and qualitative experiments were discussed. The result of these experiments is the demonstration of clear improvement of system-level performance attributable to modifying the control architecture's deliberative layer with respect to a learned model of how its plans are interpreted by the reactive layer.

CHAPTER VIII

FUTURE DIRECTIONS

8.1 Introduction

This document has focused on improving the system-level performance of hybrid control architectures for robotic systems. The proposed approach centered around defining a learning signal around the execution of spatial plans by the architecture's reactive layer. While this approach has been demonstrated to be successful in the experimental systems detailed in this document, a number of research questions have been raised or left unanswered by this work. What follows below is a non-exhaustive exploration of related open questions.

8.2 Internal Representation

While the work presented in this document was implemented on two different robotic platforms, the internal representations used did not come close to spanning the space of representations currently in use today. From the standpoint of the learning problem presented in Chapter 4, arguably, reactive layer's internal representation does not come into play. The reactive layer is treated by the learning mechanism as a "black-box", hidden by the measurement function h_c. However, the learning mechanism must make explicit use of the deliberative layer's representation of the map, M_t in creating a cost model, c_t.

Certainly, many robotic map representations are in use today [47]. The platforms presented in this document both rely on grid-based representations. The LAGR platform uses a height derivative map, built from the robot's stereo cameras. Sting platform uses an occupancy grid, built from the robot's ladar sensors. Relevant

sections of these maps are used as input to the learning mechanism to build the cost model. However, it remains an open question as to how the map representation affects the efficiency and efficacy of various learning algorithms. Because probabilistic mapping techniques have emerged as a particularly popular representation [48], an especially interesting question is how a representation of uncertainty within the map would affect learning.

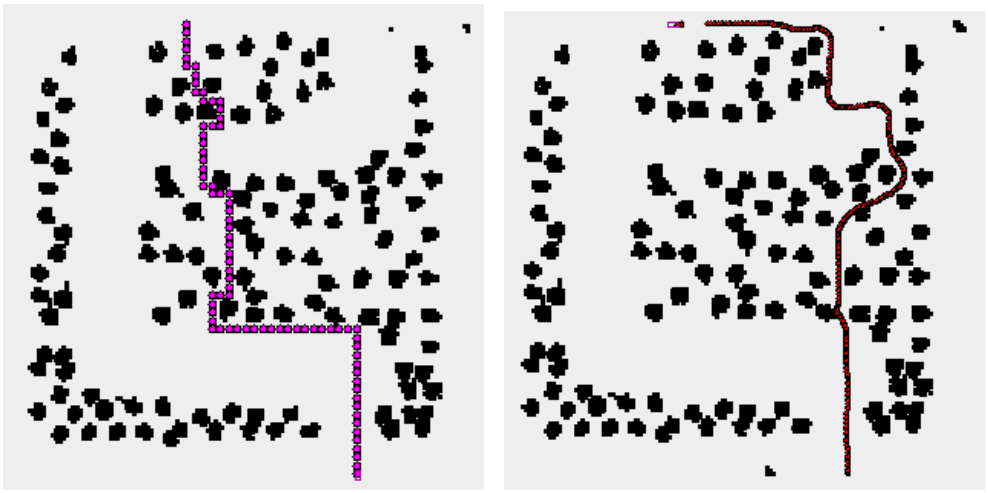
8.3 Incremental Learning

In this document, we touched on solving the problem of learning a planning cost model incrementally. By adding learning experiences to the learning mechanism as the robot explores them, the robot can change it's model "on the fly", always using the most up-to-date model as it plans. In Chapter 7, we demonstrated that as the number of experiences in a k-nearest neighbor learner increased, the system-level performance of the robot also increased. However, in this experiment the robot did not actually learn on-line. Varying numbers of experiences were batch processed, to evaluate the performance as the number of experiences varied.

This opens a line of questions pertaining to on-line or incremental learning. Obviously, questions of exploration versus exploitation arise. Early in the learning process, the robot should explore the environment, gaining as much experience as possible, to build a better model. However, at what point should it begin exploiting the model it has built? How can this trade-off be quantified? What transient effects on the robot's performance are caused by using a cost model based on only a few experiences, early in the process?

8.4 Learning the Transition Model

The results presented in Chapters 6 and 7 demonstrate that improving the planner's cost model has the potential to improve overall system performance. However,



(a) (b)

Figure 37: In (a), the planned path from the robot to the goal through the "path in the woods" environment using the baseline planner. The robot is at the bottom of the image. The goal is at the top of the image. The planned path is shown by pink waypoints. Obstacles in the map are shown in black. In (b), the actual path taken by the system (trajectory in red). Notice that the planned path takes the robot significantly to the left of the actual trajectory. The robot is unable to achieve the planned path. This inefficiency is due to a conflict between layers, pertaining to the transition model d_t.

consider the plan and trajectory shown in Figure 37. This is the baseline planning/exectution result presented in Chapter 7. Notice, the planned path takes the robot significantly, to the left of the reactively executed trajectory. This is due to a conflict between the deliberative transition model d and the reactive control. In addition to learning the cost model used by the deliberative layer, it is apparent that improving the transition model may also have a significant effect on system performance. This learning problem would be set up similarly to the cost model learning problem. The goal would be to choose a transition model d_t that is as similar as possible to the expected measured transition, h_d, given all measured transitions up to time t,

$$d_t = \arg\min_{d \in D} \; E[\; |\text{diff}(h_d(x, r^o), d(r^o, M, g))|\;] \mid \{h_d(x, r^o)\}^t$$

8.5 Learning Applied to Complex Tasks

Kinematic and dynamic constraints are not the only concepts that can be used by deliberative architectural layers to improve the system-level performance of robots. After all, many reactive behavior-based systems are designed not simply to enforce kinematic and dynamic constraints, but to perform other, more task-specific, functions. In fact, a given system may use many behavioral assemblages, each designed for a different function. In this context, planning may be viewed as sequencing configurations, in addition to navigation advice [5], [4]. Planning with respect to these complex behavioral assemblages should not be viewed merely as planning with respect to their constraints, but should be viewed as planning with respect to their capabilities.

While systems presented earlier in this work have been focused on shortest-path or fastest-path navigation, many robot navigation tasks involve more complex tasks. These tasks may, for example, include navigating within a formation [7], moving to maintain observational coverage of a target [38], or obeying a system of complex

traffic laws [52]. Sequencing behavioral assemblages to achieve specific mission-level performance goals over these tasks requires an accurate model of each behavioral assemblage's capabilities with respect to the world (i.e. a model of what the system-level behavior of each assemblage, given a model of the local environment). How can these capabilities be represented, in general, in a learning algorithm? How can qualitative goals be integrated into the framework developed in this document?

8.6 Discussion

This chapter has explored a number of open questions resulting from the work presented in this document. While results have been presented showing the effect of using learning mechanisms to minimize the conflict between layers of a hybrid control architecture on different robot platforms, it is clear that the topic is still full of avenues for future research. It is hoped that research on this topic will continue in the future, filling in the questions raised in this chapter, and others as they are raised.

CHAPTER IX

CONCLUSION

This work has dealt with the topic of improving performance in hybrid deliberative-reactive architectures in the application of spatial navigation. Noting the potential for conflict between the models used to design the respective layers, a measurement-based supervisory signal was proposed. A mechanism for learning the cost model used by the deliberative layer to evaluate potential plans was implemented. It was then shown that the deliberative layer could make use of this supervisory signal and learning mechanism to create plans that resulted in improved system-level performance when executed by the architecture's reactive layer.

This work has addressed the high-level research question posed in Chapter 1:

Research Question - In the context of spatial navigation, and the scope of the interface of the deliberative and reactive layers, how can conflict between the models of the capabilities of a robot used in the layers of a hybrid deliberative-reactive robot control architecture be detected and corrected to improve overall system-level performance?

It has also addressed the two sub-questions posed:

Sub-Question 1 - How can conflicts between the deliberative and reactive layers' kinematic and dynamic models of the robot's relationship to the world be detected?

Sub-Question 2 - How can the robot learn from these detected conflicts to change its model of its relationship to to the world, in particular within the deliberative layer?

Each of these questions has been addressed in this document. The first question was addressed in Chapters 3, 4 and 6. In Chapter 3, a conceptual supervisory signal was was defined, relating the deliberative layer's model of the effects of its output, and the measured system-level output. This signal was used as a supervisory signal to the deliberative layer. Chapter 4 formalized this approach as an optimization over the space of design of hybrid automata. In Chapter 6, this supervisory signal was implemented in terms of differing models of kinematic constraints. The deliberative layer used the signal to change its model of the robot's kinematic constraints within the world, resulting in improved performance.

The second question was addressed in Chapter 7. In Chapter 7, the supervisory signal was implemented to correct conflict between the deliberative layer's dynamical model, and the reactive layer's actual execution. The supervisory signal was used by the deliberative layer to learn a model of the cost of execution over an occupancy grid. This resulted in plans that took less time to complete and were intuitively satisfying.

In addition to posing research questions, Chapter 1 previewed several expected contributions:

- Definition, within the context of a hybrid control architecture, of inter-layer conflict, produced by the conflict between a deliberative layer's model of the effects of its commands and the outcome of a reactive layer's interpretation of those commands.

- Method to modify the deliberative layer's execution based on learned models of inter-layer conflict to improve system-level performance.

- Method to modify the deliberative layer's execution based on incrementally learned models of inter-layer conflict to improve system-level performance

- Quantitative and qualitative experimental results demonstrating the value of the above contributions in the context of robotic spatial navigation.

Each of these expected contributions has been addressed in this work. Chapters 3 and 4 defined a supervisory signal based on the reactive execution of deliberative plans, formalizing the approach in the language of hybrid control theory. Chapter 6 first demonstrated a modification of the deliberative layer's execution based on this error signal. Chapter 7 demonstrated learning algorithms applied to modifying the deliberative layer's model of the robot's execution. Experimental results were documented in Chapters 6 and 7.

It is expected that this work will serve as a significant contribution in the field of robotic control architecture design. It demonstrates a method in which conflicts between the models used by architectural layers can be corrected without significant changes to representation. Additionally, because this approach maintains the encapsulation of layers, it is expected to be very portable to different systems. As the robotics industry grows, many different complex software systems will, undoubtedly, be designed. Designs that can easily resolve conflicts between complex software modules, as this work is designed to do, will have an edge in the marketplace of ideas.

APPENDIX A

ALGORITHMS

This appendix presents pseudocode implementations of the algorithms developed for the systems referenced throughout this document. Specifically, behavior-based control, behavioral arbitration and conflict correction mechanisms are presented for both the LAGR and Sting platforms. Every effort is made to present the algorithms as they are used in the fielded systems. However, certain details, such as error checking, hardware interfaces and well-known, referenced algorithms (e.g. A* search) have not been included.

A.1 LAGR Platform

As mentioned in Chapter 5, the LAGR platform makes use of a voting-based behavior-based control strategy. The behaviors are implemented over candidate directions of travel. The *vote* data structure is used to store preferences for each candidate direction. The *vote* data structure contains three fields: $vote.\theta$, the candidate direction of travel; *vote.score*, the numerical preference for expressed for this direction; and *vote.veto*, a boolean value indicating whether the direction has been vetoed. Behavioral arbitration is done by summing the votes cast by each behavior, discounting directions that have been vetoed. Conflict correction is performed by comparing the provided planning waypoint with the set of vetoed candidate directions. The *params* data structure is used to hold any parameters that are set at configuration time.

A.1.1 Utility Functions

CAST-GAUSSIAN-DIRECTIONAL-VOTE **-** indicates a preference for a direction over the array of candidate directions, *vote-array*, by distributing votes according to

a normal Gaussian function, with mean, θ-*mean* and standard deviation, θ-*std*.

CAST-GAUSSIAN-DIRECTIONAL-VOTE(*vote-array*, θ-*mean*, θ-*std*, *weight*)

```
1  for each vote ∈ vote-array
2       do
3            θ-diff ← abs(θ-mean − vote.θ)
4            vote.score ← normalPDF(0, θ-std, θ-diff)
             ▷ normalPDF takes as arguments mean, std, value
5  return
```

NORMALIZE-SCORES **-** scales the *score* field of each vote such that the sum of the scores over the votes in the array is equal to 1.0.

NORMALIZE-SCORES(*vote-array*)

```
1  score-sum ← 0
2  for each vote ∈ vote-array
3       do
4            score-sum ← score-sum + vote.score
5  for each vote ∈ vote-array
6       do
7            vote.score ← vote.score / score-sum
8  return
```

A.1.2 Behavioral Arbiter

ARBITRATE-DIRECTIONAL-VOTES **-** chooses a direction in which to travel by arbitrating between all the active behaviors. The behaviors' votes are provided as an array of arrays of vote structures, *vote-array-array*. An array of weights, *weights* provides a respective weight for each behavior's input. When summing

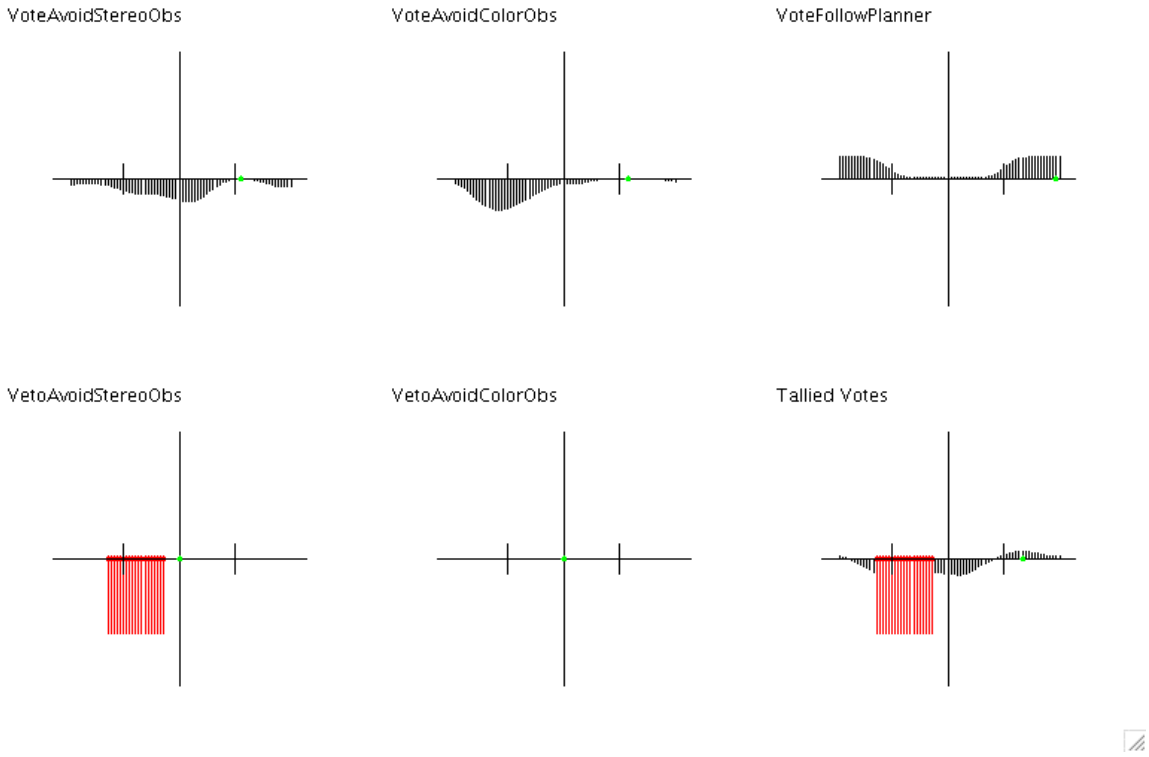


Figure 38: A graphical representation of the voting scheme employed to navigate the robot. The x-axis of each plot represents an ego-centric angular distribution of possible paths around the robot in the range $(-\pi, +\pi]$, with 0 being in front of the robot. The y-axis represents the relative preference of each path, according to the respective controller. Vetoes are drawn as large negative values. The last plot represents the sum of the votes provided by all the controllers. The largest non-vetoed value is chosen for action by the robot. In this example, the first behavior resists a stereo-perceived obstacles to the front and left of the robot. A color-based obstacle is perceived to the left. The plan tells the robot to go backwards, and the left is vetoed as a result of stereo obstacles. The tallied votes tell the robot to go to the right.

the votes, each vote is weighted by its associated behavior's weight. Vetoes are "summed" by executing an inclusive OR over all the vetoes. The output direction is chosen by selecting the direction with the largest score, that has not been vetoed.

```
ARBITRATE-DIRECTIONAL-VOTES(vote-array-array, weights)
 1  vote-array-sum ← CREATE-ZEROED-VOTE-ARRAY()
 2  for each i ← 0 . . . vote-array-array.length
 3        do
 4           vote-array-sum ← ADD-VOTES(vote-array-sum,
 5           vote-array-array[i], weights[i])
 6  max-score ← −∞
 7  max-score-vote ← NULL
 8  for each vote ∈ vote-array-sum
 9        do
10           if ! vote.veto & vote.score > max-score
11              then
12                   max-score ← vote.score
13                   max-score-vote ← vote
14  if max-score-vote = NULL
15  ▷ All votes are vetoed. Take the vote with the highest score.
16     then
17           for each vote ∈ vote-array-sum
18                do
19                   if vote.score > max-score
20                      then
21                           max-score ← vote.score
22                           max-score-vote ← vote
23  output-θ ← max-score-vote.θ
24  return output-θ
```

A.1.3 Behaviors

MOVE-TO-WAYPOINT **-** expresses a preference for moving toward the provided planning waypoint by distributing votes as a Gaussian function around the direction to the waypoint.

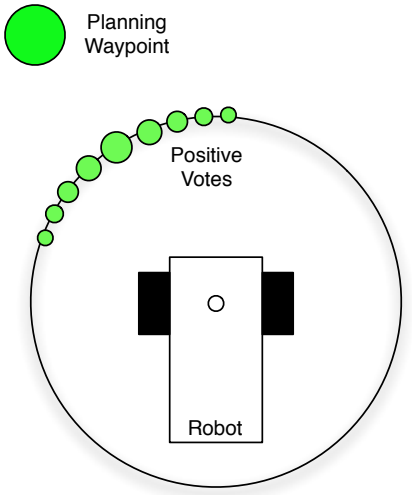


Figure 39: The *move to waypoint* behavior.

MOVE-TO-WAYPOINT(*vote-array*, *waypoint*)

1 *angle* ← atan2(*waypoint.y*, *waypoint.x*)

2 CAST-GAUSSIAN-DIRECTIONAL-VOTE(*vote-array*, *angle*, *params.θ-std*, *1.0*)

3 NORMALIZE-SCORES(*vote-array*)

4 **return**

AVOID-STEREO-OBSTACLES **-** places negative votes against the direction of any stereo vision-detected obstacles. Obstacles that are closer to the robot generate stronger negative votes.

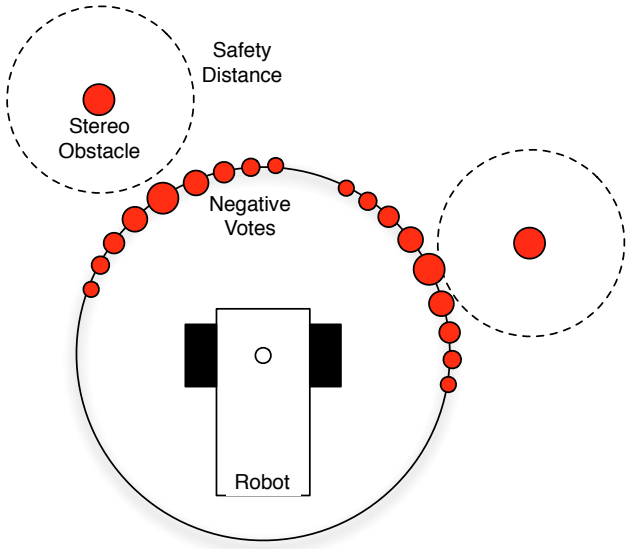


Figure 40: The *avoid stereo obstacles* behavior.

AVOID-STEREO-OBSTACLES(*vote-array*, *stereo-obstacle-array*)

1 **for** each *obstacle* ∈ *stereo-obstacle-array*
2 **do**
3 *angle* ← atan2(*obstacle.y*, *obstacle.x*)
4 *dist* ← distance(*obstacle*)
5 *weight* ← max(*params.min-weight*, $-\ dist^{-2}$)
6 CAST-GAUSSIAN-DIRECTIONAL-VOTE(*vote-array*, *angle*, *params.stereo-θ-std*, *weight*)
7 NORMALIZE-SCORES(*vote-array*)
8 **return**

VETO-STEREO-OBSTACLES **-** vetoes any direction which will cause a collision with a stereo vision-detected obstacle within a parameterizable safety distance.

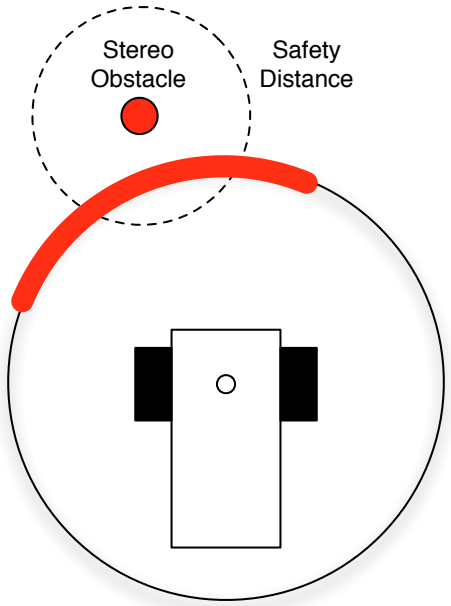


Figure 41: The *veto stereo obstacles* behavior.

VETO-STEREO-OBSTACLES(*vote-array*, *stereo-obstacle-array*)

```
1  for each obstacle ∈ stereo-obstacle-array
2       do
3          angle ← atan2(obstacle.y, obstacle.x)
4          dist ← distance(obstacle)
5          if dist ≤ params.safety-dist
6             then
7                  width ← 2 acos(params.robot-width / (2 dist))
8                  for each vote ∈ vote-array
9                       do
10                        if vote.θ − angle < width/2
11                          then
12                               vote.veto ← TRUE
13 return
```

AVOID-COLOR-OBSTACLES - places negative votes against the direction of any color vision-detected obstacles. Obstacles that are closer to the robot generate stronger negative votes.

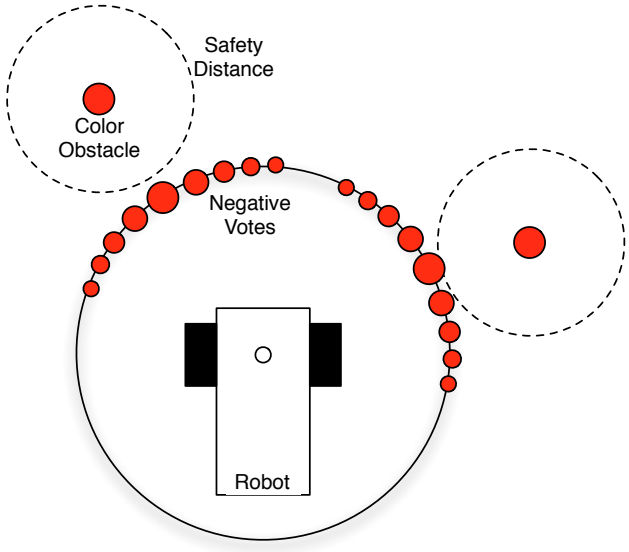


Figure 42: The *avoid color obstacles* behavior.

AVOID-COLOR-OBSTACLES(*vote-array*, *color-obstacle-array*)

1 **for** each *obstacle* $\in$ *color-obstacle-array*
2 **do**
3 *angle* $\leftarrow$ atan2(*obstacle.y*, *obstacle.x*)
4 *dist* $\leftarrow$ distance(*obstacle*)
5 *weight* $\leftarrow$ max(*params.min-weight*, $-\ dist^{-2}$)
6 CAST-GAUSSIAN-DIRECTIONAL-VOTE(*vote-array*, *angle*, *params.color-θ-std*, *weight*)
7 NORMALIZE-SCORES(*vote-array*)
8 **return**

VETO-COLOR-OBSTACLES - vetoes any direction which will cause a collision with a color vision-detected obstacle within a parameterizable safety distance.

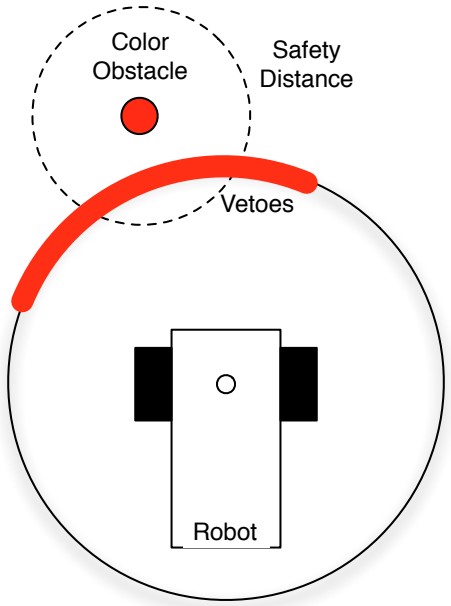


Figure 43: The *veto color obstacles* behavior.

VETO-COLOR-OBSTACLES(*vote-array*, *color-obstacle-array*)

```
1  for each obstacle ∈ color-obstacle-array
2       do
3          angle ← atan2(obstacle.y, obstacle.x)
4          dist ← distance(obstacle)
5          if
6          dist ≤ params.safety-dist
7            then
8                 width ← 2 acos(params.robot-width / (2 dist))
9                 for each vote ∈ vote-array
10                     do
11                        if vote.θ − angle < width/2
12                          then
13                               vote.veto ← TRUE
14 return
```

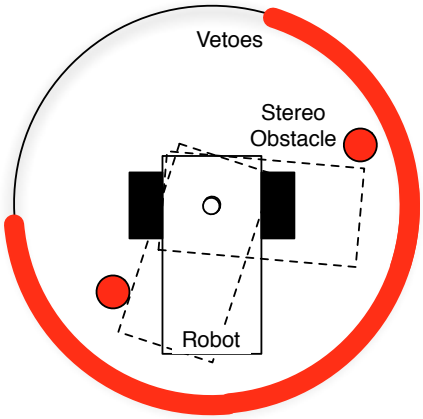


Figure 44: The *veto dangerous obstacles* behavior.

VETO-DANGEROUS-TURNS - casts vetoes against any direction which cannot be achieved by turning in place, due to obstacles blocking the path of robot over the proposed rotation.

```
VETO-DANGEROUS-TURNS(vote-array, stereo-obstacle-array)
 1  collision-left ← ∞
 2  collision-right ← −∞
 3  for each obstacle ∈ stereo-obstacle-array
 4        do
 5           for each vote ∈ vote-array
             ▷ Evaluate turning left
 6           temp − obstacle ← rotate-vector(obstacle, vote.θ)
 7                 do
 8                    if temp-obstacle.x < params.robot-front-x &
                      obstacle.x > params.robot-back-x &
                      abs(obstacle.y − (params.robot-width)/2)
                      ▷ If this angle brings the robot into
                      ▷ collision with the obstacle
 9                      then
10                           if vote.θ < collision − left
11                              then
12                                   collision-left ← vote.θ
13                           if vote.θ > collision − right
14                              then
15                                   collision-right ← vote.θ
16  for each vote ∈ vote-array
17        do
18           if vote.θ > collision-left & vote.θ < collision-right
19              then
20                   vote.veto ← TRUE
21  return
```

A.1.4 Kinematic Conflict Correction

REACTIVE-LOOP - performs the functions of the reactive layer of the architecture. First, sensory information is updated, integrated, and communicated to the deliberative layer. A new plan, if available is received from the deliberative layer. Then, the behaviors are run, using the most recent planning and sensory information. The behavioral arbitrator then chooses a direction in which to travel, which is executed by the low-level motor control. Finally, a summary of the vetoes cast by the behaviors is sent to the conflict correction module.

REACTIVE-LOOP()

1 UPDATE-LOW-LEVEL-SENSING-MODULES()
2 UPDATE-INTEGRATED-SENSING-MODULES()
3 SEND-SENSORY-DATA-TO-DELIBERATIVE-MODULE(*robot-pose*, *stereo-obstacles*, *color-obstacles*)
4 RECEIVE-PLAN-FROM-DELIBERATIVE-MODULE()
5 *vote-array-array* ← CREATE-VOTE-ARRAY-ARRAY
6 MOVE-TO-WAYPOINT(*vote-array-array*[0], *plan-waypoint*)
7 AVOID-STEREO-OBSTACLES(*vote-array-array*[1], *stereo-obstacles*)
8 VETO-STEREO-OBSTACLES(*vote-array-array*[2], *stereo-obstacles*)
9 AVOID-COLOR-OBSTACLES(*vote-array-array*[3], *color-obstacles*)
10 VETO-COLOR-OBSTACLES(*vote-array-array*[4], *color-obstacles*)
11 *commanded-θ* ← ARBITRATE-DIRECTIONAL-VOTES(*vote-array-array*, *params.weights*)
12 ACTUATE-MOTORS(*commanded-θ*)
▷ ACTUATE-MOTORS controls the robot's velocity from a low level
13 SEND-VETOS-TO-CONLICT-CORRECTION-MODULE(*vote-array-array*[2])
14 **return**

DELIBERATIVE-LOOP **-** performs the functions of the deliberative layer of the architecture. First, sensory data is received from the reactive layer, which is then integrated into the map. Then input from the conflict correction module is received and integrated into the map. A* search is then used to create a plan over the map, which is sent to the reactive layer and conflict correction module.

DELIBERATIVE-LOOP(*goal*)

1 (*stereo-obstacles*, *color-obstacles*) ←
RECEIVE-SENSORY-DATA-FROM-REACTIVE-MODULE()
2 INTEGRATE-SENSORY-DATA-INTO-MAP(*stereo-obstacles*,
color-obstacles, *map*)
3 *conflict-constraints* ←
RECEIVE-CONSTRAINTS-FROM-CONFLICT-CORRECTION-MODULE()
4 INTEGRATE-CONSTRAINTS-INTO-MAP(*conflict-constraints*, *map*)
5 *plan* ← PLAN-TO-GOAL-A*(*goal*, *map*)
6 SEND-PLAN-TO-REACTIVE-MODULE(*plan*)
7 SEND-PLAN-TO-CONFLICT-CORRECTION-MODULE(*plan*)
8 **return**

CONFLICT-CORRECTION-LOOP **-** compares the output of the reactive and deliberative layers, sending input to the deliberative layer when the two are in conflict. The list of vetoed directions and the next planning waypoint are received from the reactive and deliberative layers, respectively. If the direction of the waypoint has been vetoed, a constraint is sent to the deliberative layer, preventing it from planning through that point.

```
CONFLICT-CORRECTION-LOOP()
 1  vote-array ← RECEIVE-VETOS-FROM-REACTIVE-LOOP()
 2  plan-waypoint ← RECEIVE-PLAN-FROM-DELIBERATIVE-LOOP()
 3  angle ← atan2(plan-waypoint.y, plan-waypoint.x)
 4  constraints ← CREATE-EMPTY-CONSTRAINTS-SET()
 5  for each vote ∈ vote-array
 6       do
 7          if vote.veto & vote.θ − angle ≤ params.angle-threshold
 8             then
 9                  ADD-CONSTRAINT-LOCATION(constraints,
                    plan-waypoint)
10  if !EMPTY(constraints)
11     then
12          SEND-CONTRAINTS-TO-DELIBERATIVE-LAYER(constraints)
13
14  return
```

A.2 Sting Platform

Like the LAGR platform, the Sting platform makes use of a voting-based behavior-based control strategy. The behaviors are implemented over candidate curvature arcs of travel (which corresponds to steering angle in an Ackermann steering model). As with the LAGR platform, the *params* data structure is used to hold any parameters that are set at configuration time, and the *vote* data structure is used to store preferences for each candidate direction. The *vote* data structure contains three fields: $vote.\kappa$, the candidate curvature of travel; $vote.v$, the maximum allowable velocity along the associated curvature arc; and *vote.score*, the numerical preference for expressed for this direction. The default value for $vote.v$ is $params.maxV$, the maximum

allowable velocity under any circumstance. Behavioral arbitration is done by summing the votes cast by each behavior, choosing the curvature arc associated with the highest summed score. A velocity set-point is chosen by selecting the minimum of the velocities allowed along the selected arc. Conflict correction is performed by learning a model of the average velocity of the reactive execution of prescribed plans.

A.2.1 Utility Functions

TRANSFORM-LINEARLY **-** performs a linear transformation on an input value.

TRANSFORM-LINEARLY(*input*, *input-min*, *input-max*, *output-min*, ´*output-max*)

1 $output \leftarrow \frac{(input - input\text{-}min)(output\text{-}max - output\text{-}min)}{(input\text{-}max - input\text{-}min)} + output\text{-}min$
2 **if** *params.saturate-transform*
3 **then**
4 $output \leftarrow \max(output, output\text{-}min)$
5 $output \leftarrow \min(output, output\text{-}max)$
6 **return** *output*

TRANSFORM-PIECEWISE-LINEARLY **-** performs a piecewise-linear transformation on an input value.

TRANSFORM-PIECEWISE-LINEARLY(*input*, *input-min-array*, ´*input-max-array*, *output-min-array*, *output-max-array*)

```
1  i ← 0
2  while !(input-min-array[i] < input < input-max-array[i])
     ▷ Needless to say, safety checks are necessary in order to
     ▷ assume this relation will be satisfied
3        do
4           i ← i + 1
5  input-min ← input-min-array[i]
6  input-max ← input-max-array[i]
7  output-min ← output-min-array[i]
8  output-max ← output-max-array[i]
9  output ← ((input − input-min)(output-max − output-min)) / (input-max − input-min) + output-min
10 return output
```

CAST-GAUSSIAN-CURVATURE-VOTE **-** indicates a preference for a curvature over the array of candidate arcs, *vote-array*, by distributing votes according to a normal Gaussian function, with mean, κ-*mean* and standard deviation, κ-*std*.

```
CAST-GAUSSIAN-CURVATURE-VOTE(vote-array, κ-mean, κ-std, weight)
1  for each vote ∈ vote-array
2        do
3           κ-diff ← abs(κ-mean − vote.κ)
4           vote.score ← normalPDF(0, κ-std, κ-diff)
              ▷ normalPDF takes as arguments mean, std, value
5  return
```

NORMALIZE-SCORES **-** scales the *score* field of each vote such that the sum of the scores over the votes in the array is equal to 1.0.

NORMALIZE-SCORES(*vote-array*)

1 *score-sum* $\leftarrow 0$

2 **for** each *vote* $\in$ *vote-array*

3 **do**

4 *score-sum* $\leftarrow$ *score-sum* + *vote.score*

5 **for** each *vote* $\in$ *vote-array*

6 **do**

7 *vote.score* $\leftarrow \frac{vote.score}{score\text{-}sum}$

8 **return**

A.2.2 Behavioral Arbiter

ARBITRATE-CURVATURE-VELOCITY-VOTES **-** chooses a curvature arc along which to travel by arbitrating between all the active behaviors. The behaviors' votes are provided as an array of arrays of vote structures, *vote-array-array*. An array of weights, *weights* provides a respective weight for each behavior's input. When summing the votes, each vote is weighted by its associated behavior's weight. The output curvature is chosen by selecting the direction with the largest score. The output velocity is chosen by selecting the minimum allowable velocity along the chosen curvature.

```
ARBITRATE-CURVATURE-VELOCITY-VOTES(vote-array-array, weights)
 1  vote-array-sum ← CREATE-ZEROED-VOTE-ARRAY()
 2  for each vote-array ∈ vote-array-array
 3       do
 4            vote-array-sum ← ADD-VOTES(vote-array-sum, vote-array,
              weights)
              ▷ ADD-VOTES returns a vote-array containing the
              ▷ respectively weighted summed vote scores
 5  max-score-vote ← ARGMAX-BY-SCORE(vote-array-sum)
 6  output.κ ← max-score-vote.κ
 7  output.v ← ∞
 8  for each vote-array ∈ vote-array-array
 9       do
10            for each vote ∈ vote-array
11                 do
12                      if vote.κ = output.κ
13                        then
14                                output.v ← min(output.v, vote.v)
15  return (output.κ, output.v)
```

A.2.3 Behaviors

MOVE-TO-WAYPOINT **-** expresses a preference for moving toward the provided planning waypoint by distributing votes as a Gaussian function around the mean curvature, chosen by a linear controller based on the angle to the waypoint.

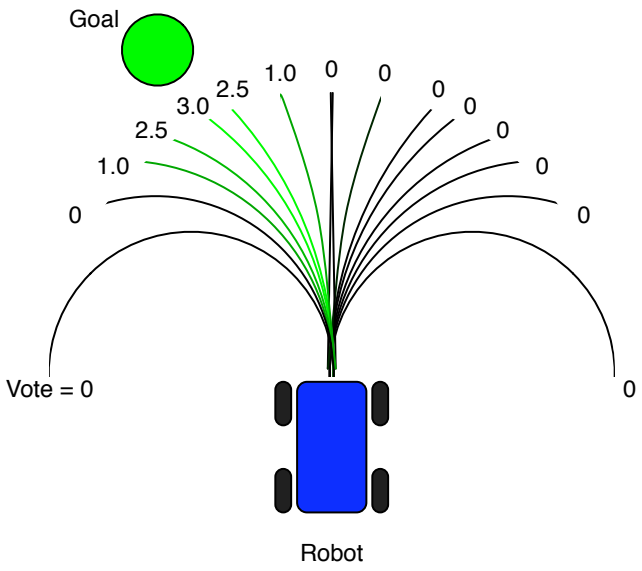


Figure 45: The *move to waypoint* behavior.

MOVE-TO-WAYPOINT(*vote-array*, *waypoint*)

1 *angle* ← atan2(*waypoint.y*, *waypoint.x*)

2 *kMean* ← TRANSFORM-LINEARLY(*angle*, − *params.angleSaturation*, *params.angleSaturation*, − *params.curvatureSaturation*, *params.curvatureSaturation*)

3 CAST-GAUSSIAN-CURVATURE-VOTE(*vote-array*, *kMean*, *params.kstd*, *1.0*)

4 NORMALIZE-SCORES(*vote-array*)

5 **return**

AVOID-OBSTACLES - places negative votes against arcs that come within a minimum distance of an obstacle. Obstacles that are closer to the robot along the arc generate stronger negative votes.

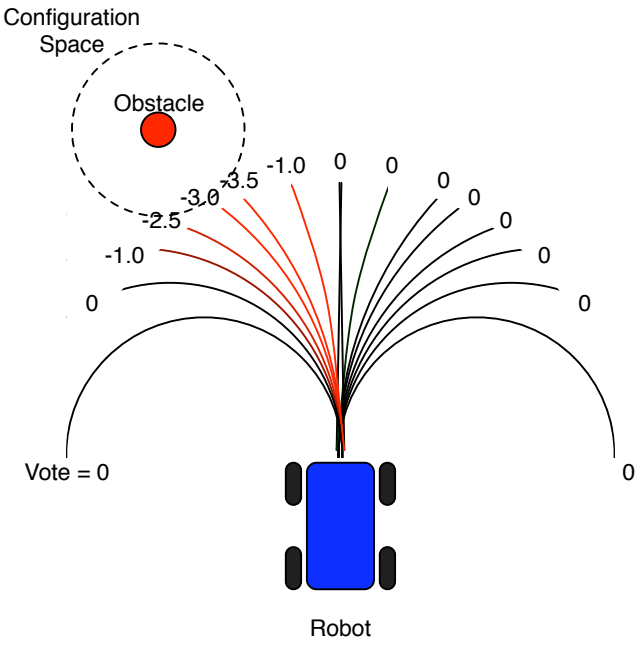


Figure 46: The *avoid obstacles* behavior.

AVOID-OBSTACLES(*vote-array*, *obstacle-array*, *robot-pose*)

```
1  for each vote ∈ vote-array
2       do
3          arc-radius ← inv vote.κ
4          closest-obstacle-dist ← ∞
5          arc-center ← FIND-ARC-CENTER(robot-pose, vote.κ)
           ▷ FIND-ARC-CENTER finds the center of the circle defined
           ▷ by the curvature and the robot's current pose
6          for each obstacle ∈ obstacle-array
7               do
8                  obstacle-dist ← abs(distance(arc-center, obstacle)
                   − arc-radius)
9                  closest-obstacle-dist ← min(closest-obstacle-dist,
                   obstacle-dist)
10         if closest-obstacle-dist ≤ params.obstacle-relevance-dist
11           then
12                 vote-weight ← −params.weight-const / closest-obstacle-dist^params.weight-pow
13                 CAST-GAUSSIAN-CURVATURE-VOTE(vote.κ,
                   params.κ-std, vote-weight)
14 NORMALIZE-SCORES(vote-array)
15 return
```

MAINTAIN-HEADWAY **-** restricts the allowable velocity along arcs that come within a minimum distance of an obstacle. Obstacles that are closer to the robot along the arc generate slower allowable velocities, eventually to a stop.

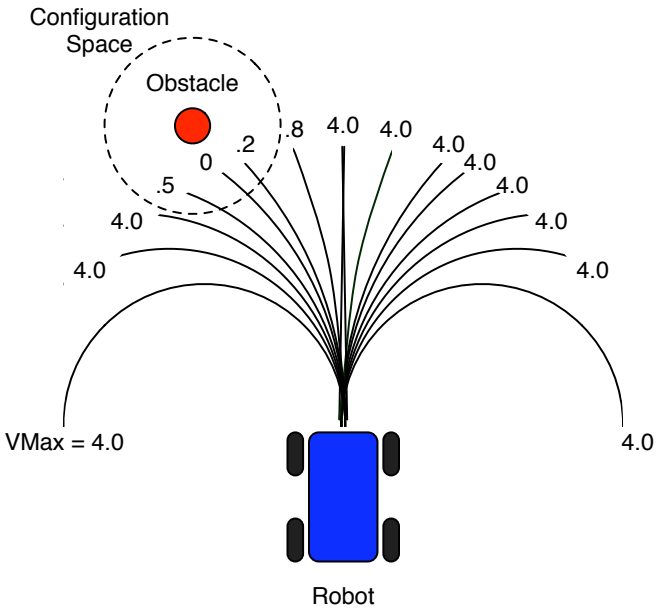


Figure 47: The *maintain headway* behavior.

```
MAINTAIN-HEADWAY(vote-array, obstacle-array, robot-pose)
 1  for each vote ∈ vote-array
 2       do
 3          arc-radius ← inv vote.κ
 4          closest-collision-dist ← ∞
 5          arc-center ← FIND-ARC-CENTER(robot-pose, vote.κ)
 6          for each obstacle ∈ obstacle-array
 7               do
 8                    collision-dist ← FIND-DIST-TO-COLLISION(vote.κ,
                      obstacle, robot-pose, params.lateral-config-space)
                      ▷ FIND-DIST-TO-COLLISION finds the distance along
                      ▷ the arc before collision with the obstacle,
                      ▷ using the given lateral configuration space
 9                    closest-collision-dist ← min(closest-collision-dist,
                      collision-dist)
10          if closest-collision-dist ≤ params.collision-relevance-dist
11            then
12                    vote.v ← TRANSFORM-PIECEWISE-LINEARLY(
                      closest-collision-dist, params.collision-dist-max-array,
                      params.collision-dist-max-array,
                      params.speed-min-array, params.speed-max-array)
13  NORMALIZE-SCORES(vote-array)
14  return
```

SLOW-FOR-CONGESTED-AREAS **-** restricts the allowable velocity along arcs that come within the general area of an obstacle. Obstacles that are closer to the robot along the arc generate slower allowable velocities.

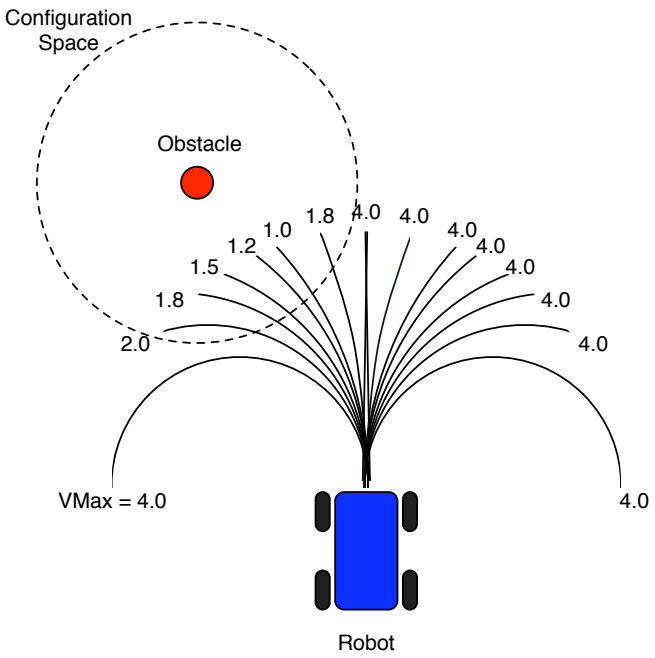


Figure 48: The *slow for congested areas* behavior.

```
SLOW-FOR-CONGESTED-AREAS(vote-array, obstacle-array)
 1  for each vote ∈ vote-array
 2       do
 3          arc-radius ← inv vote.κ
 4          closest-collision-dist ← ∞
 5          arc-center ← FIND-ARC-CENTER(robot-pose, vote.κ)
 6          for each obstacle ∈ obstacle-array
 7               do
 8                    collision-dist ← FIND-DIST-TO-COLLISION(vote.κ,
                      obstacle, robot-pose, params.large-lateral-config-space)
                      ▷ Note the larger configuration space than used in
                      ▷ MAINTAIN-HEADWAY
 9                   closest-collision-dist ← min(closest-collision-dist,
                      collision-dist)
10          if closest-collision-dist ≤ params.collision-relevance-dist
11            then
12                   vote.v ← TRANSFORM-LINEARLY(closest-collision-dist,
                      params.slow-min-dist, params.slow-relevance-dist,
                      params.min-speed, params.max-speed)
13  NORMALIZE-SCORES(vote-array)
14  return
```

SLOW-FOR-TURNS - restricts the allowable velocity along arcs according to a maximum allowable rotational velocity.

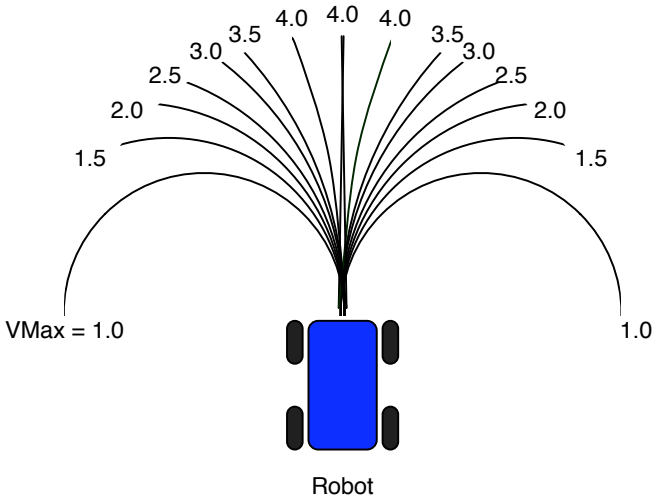


Figure 49: The *slow for turns* behavior.

SLOW-FOR-TURNS(*vote-array*)

1 **for** each $vote \in vote\text{-}array$

2 **do**

3 $vote.v \leftarrow \min(params.max\text{-}speed, \frac{params.max\text{-}rotational\text{-}vel}{\text{abs}(vote.\kappa)})$

4 NORMALIZE-SCORES(*vote-array*)

5 **return**

A.2.4 Dynamical Conflict Correction

REACTIVE-LOOP **-** performs the functions of the reactive layer of the architecture. First, sensory information is updated, integrated, and communicated to the deliberative layer. A new plan, if available is received from the deliberative layer. Then, the behaviors are run, using the most recent planning and sensory information. The behavioral arbitrator then chooses a curvature and velocity, which are executed by the low-level motor control.

```
REACTIVE-LOOP()
 1  UPDATE-LOW-LEVEL-SENSING-MODULES()
 2  UPDATE-INTEGRATED-SENSING-MODULES()
 3  SEND-SENSORY-DATA-TO-
    DELIBERATIVE-MODULE(robot-pose, ladar-obstacles)
 4  RECEIVE-PLAN-FROM-DELIBERATIVE-MODULE()
 5  vote-array-array ← CREATE-VOTE-ARRAY-ARRAY
 6  MOVE-TO-WAYPOINT(vote-array-array[0], plan-waypoint)
 7  AVOID-OBSTACLES(vote-array-array[1], ladar-obstacles)
 8  MAINTAIN-HEADWAY(vote-array-array[2], ladar-obstacles)
 9  SLOW-FOR-CONGESTED-AREAS(vote-array-array[3], ladar-obstacles)
10  SLOW-FOR-TURNS(vote-array-array[4])
11  (commanded-κ, commanded-v) ←
    ARBITRATE-CUVATURE-VELOCITY-VOTES(vote-array-array,
    params.weights)
12  ACTUATE-MOTORS(commanded-κ, commanded-v)
13  return
```

DELIBERATIVE-LOOP - performs the functions of the deliberative layer of the architecture. First, sensory data is received from the reactive layer, which is then integrated into the map. Then, if available, a new planning cost function is received from the conflict correction module. D* search is then used to create a plan over the map, which is sent to the reactive layer. The plan and the relevant segment of the map is sent to the conflict correction module.

DELIBERATIVE-LOOP(*goal*)

1 (*stereo-obstacles*, *color-obstacles*) ←
RECEIVE-SENSORY-DATA-FROM-REACTIVE-MODULE()
2 INTEGRATE-SENSORY-DATA-INTO-MAP(*stereo-obstacles*,
color-obstacles, *map*)
3 *cost-function* ←
RECEIVE-COST-FUNCTION-FROM-CONFLICT-CORRECTION-MODULE()
4 *plan* ← PLAN-TO-GOAL-D*-LITE(*goal*, *map*, *cost-function*)
5 SEND-PLAN-TO-REACTIVE-MODULE(*plan*)
6 SEND-PLAN-TO-CONFLICT-CORRECTION-MODULE(*plan*)
7 SEND-MAP-SECTION-TO-CONFLICT-CORRECTION-MODULE(*map*)
8 **return**

CONFLICT-CORRECTION-LOOP **-** learns a model of the reactive execution of the deliberative layer's plans, expressing it as a planning cost function. First it receives a representation of the plan and map from the deliberative layer. It then measures the robot's system-level performance over the next segment of the plan. This measurement is then integrated into a k-nearest-neighbor regression model, predicting performance (average velocity), given a relative waypoint and a local segment of the map. This cost function is then sent to the deliberative layer for execution.

```
CONFLICT-CORRECTION-LOOP()
 1  plan-waypoint ← RECEIVE-PLAN-FROM-DELIBERATIVE-LOOP()
 2  relative-waypoint ← plan-waypoint − last-waypoint
 3  map-section ← RECEIVE-MAP-SECTION-FROM-DELIBERATIVE-LOOP()
 4  avg-velocity ←
    MEASURE-VELOCITY-TOWARD-WAYPOINT(plan-waypoint)
    ▷ This function blocks until the waypoint is achieved
 5  ADD-TO-EXPERIENCE-SET(experience-set,
    (avg-velocity, relative-waypoint, map-section))
 6  if params.update-incrementally
 7     then
 8          cost-function ←
            CREATE-COST-FUNCTION-kNN(experience-set)
 9          SEND-COST-FUNCTION-TO-
            DELIBERATIVE-MODULE(cost-function)
10     else
    ▷ Batch Processing
11     then
12          if FINISHED-TRAINING()
13            then
14                cost-function ←
                  CREATE-COST-FUNCTION-kNN(experience-set)
15                SEND-COST-FUNCTION-TO-
                  DELIBERATIVE-MODULE(cost-function)
            ▷ Else do nothing
16
17  return
```

APPENDIX B

EXPERIMENTAL DATA

This appendix presents compiled raw and analyzed data from the experiments described in this document. The raw data is presented as collected, paired with the corresponding statistical analysis.

B.1 Kinematic Experiments

The experiments referenced in Chapter 6 involved a robot navigating out of a cul-de-sac, using stereo vision, color vision, and a GPS signal. The cul-de-sac contained a narrow, non-navigable opening at its bottom. A reactive behavior-based controller guided the robot, based on immediate sensory perception, using a three-dimensional (x, y, θ) configuration space. A deliberative global path planner provided input based on a map built from integrated sensory information, using a two-dimensional configuration space. A conflict correction module detected and corrected conflict between the two layers. Use of the conflict correction module was shown to improve the robot's performance in negotiating the cul-de-sac.

Experiments were run using four different configurations; using a global planner only, reactive controllers only, a global planner in conjunction with reactive controllers, and a global planner in conjunction with reactive controllers and a conflict correction module. Figure 50 shows the resulting maps and plans, created by the deliberative layer, along with the reactively executed trajectory for each run.

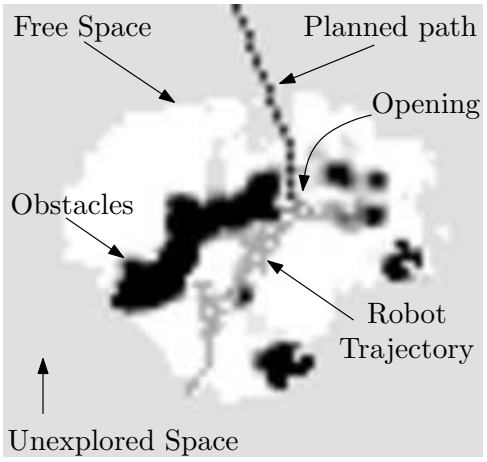
Global Planner Only - In the first run, only the planner was affecting the motion of the robot, and the only active low-level controller was a plan-following controller. As was to be expected, the planner found the opening and tried to push

through (Figure 50(a)), with the result that the robot crashed into one of the logs defining the boundary of the opening.

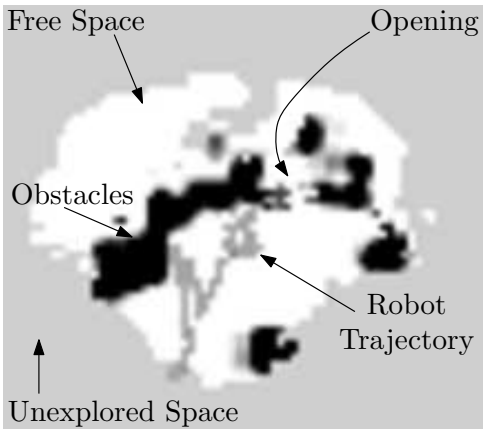
Reactive Controllers Only - In the second run, only the reactive, local controllers were active, and no global plan was provided from the planner. This control strategy exhibited the well-known and expected behavior of getting stuck in the cul-de-sac without any global information (aside from the heading to the goal) to guide the robot (Figure 50(b)).

Global Planner with Reactive Controllers - In this scenario, the planner continued to plan through the narrow opening. Meanwhile, the reactive controllers vetoed that action. As a result, the robot did not exhibit any improved behavior over the reactive-controller-only situation (Figure 50(c)).

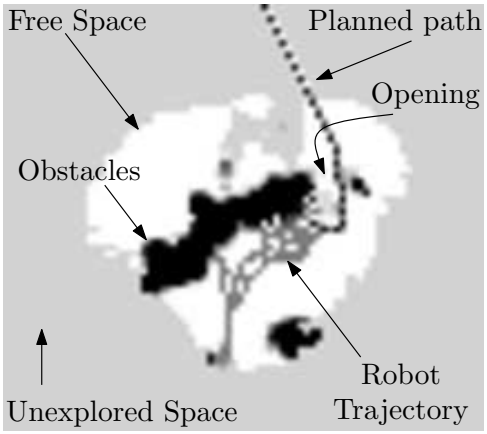
Global Planner with Reactive Controllers and Conflict Resolution - Here, the planner once again tried to force the robot through the opening in the cul-de-sac wall. However, the reactive-controller vetoed this action as well as encoded this veto through the conflict correction mechanism as a constraint in the map, and the planner then re-planned its course of action. As seen in Figure 22(d), after a bit of exploring of the cul-de-sac, the robot decided that there was no way forward through the cul-de-sac, and a path was planned out from the area, which enabled the robot to continue its mission.



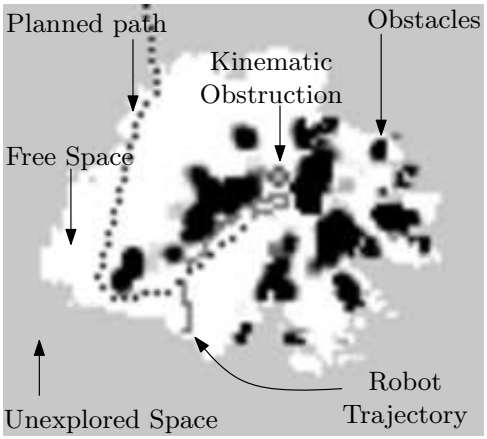
(a) The map, trajectory and plan resulting from an experiment using only a planner. Because the planner is too optimistic for the configuration space of the physical robot, the robot collides with obstacles while trying to navigate through the narrow opening.



(b) The map and trajectory resulting from an experiment using only reactive controllers. The safety-minded controllers kept the robot a safe distance from all obstacles, but did not allow progress to the goal location.



(c) The map, trajectory and plan resulting from an experiment using a planner which influences reactive controllers. The optimistic planner guides the robot toward the narrow opening, while the safety-minded controllers prevent the robot from entering. The result is that the robot loiters around the mouth of the opening.



(d) The map, trajectory and plan resulting from an experiment using a planner which influences reactive controllers with a correction signal back to the deliberative layer. The planner initially guides the robot toward the narrow opening, but the reactive controllers veto this action, noting that action in the global map. Using this information, the planner finds a path through the only safe opening in the Culde-sac.

Figure 50: Trajectories and maps created by the four experimental systems.

B.2 Dynamical Experiments

The experiments referenced in Chapter 7 involved a simulated car-like robot navigating the environments shown in Figure 51. A reactive behavior-based controller guided the robot, based on immediate ladar-based perception, using a three-dimensional (x, y, θ) configuration space. A deliberative global path planner provided input based on a map built from integrated sensory information, using a two-dimensional configuration space. A conflict correction module learned models of the reactive execution of the deliberative layer's plans. These plans were used by the deliberative layer, in the form of improved planning cost functions, to improve the robot's system level performance. Use of the conflict correction module was shown to improve the robot's performance in negotiating the cul-de-sac.

Table 10: Results of a 10-fold cross validation test on several learners predicting the robot's velocity given a proposed planning transition. The k-Nearest Neighbor algorithm with the k-value set to 5 produced the best results, nearly cutting the average relative error in half, compared to the baseline approach.

	Baseline	KNN $k = 50$	KNN $k = 15$	KNN $k = 5$	KNN $k = 2$	ANN	SVM Reg.	Lin. Reg.
Average Relative Error	106%	80%	72%	64%	61%	91%	77%	86%
Correlation	-.44	.57	.62	.65	.63	.55	.47	.50

Training experiences used to learn the planning cost function were learned from executing a random walk around the environment shown in Figure 51(a). 5000 training experiences were collected. 10-fold cross validation tests were performed on this data, using a variety of machine learning techniques. This analysis is shown in Table 10. The k-nearest-neighbor algorithm with $k = 5$ was chosen for use in further experiments.

Three major experiments were performed. The first compared the quantitative

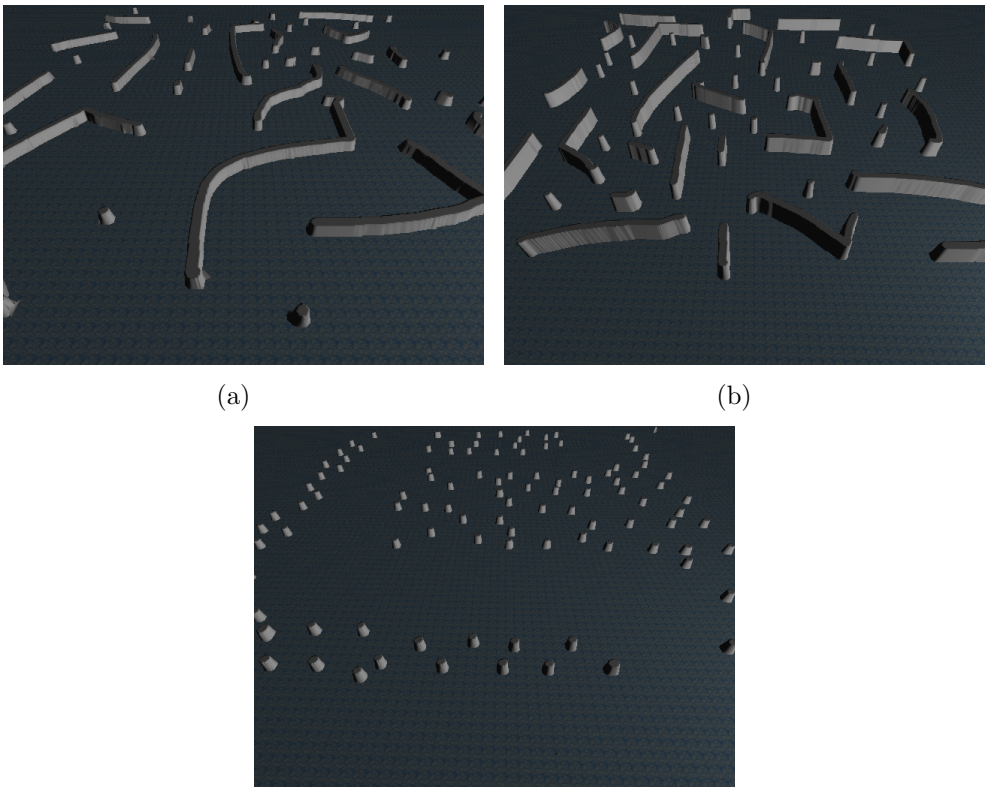


Figure 51: The three simulation environments used in testing. (a) is the training environment. (b) is the quantitative testing environment. (c) is the qualitative testing environment.

performance of a system using a planning cost function learned from the training experiences to a baseline system. The second compared the quantitative performance of the experimental system given varying numbers of training examples. The third experiment compared the qualitative performance of the experimental system and the baseline system. The first and second experiments were run in the environment shown in Figure 51(b). The robot was tasked with executing the goals shown in Figure 52 in order while the robot's performance was measured. The third experiment was run in the environment shown in Figure 51(c). The robot was tasked with traveling from one side of the environment to the other.

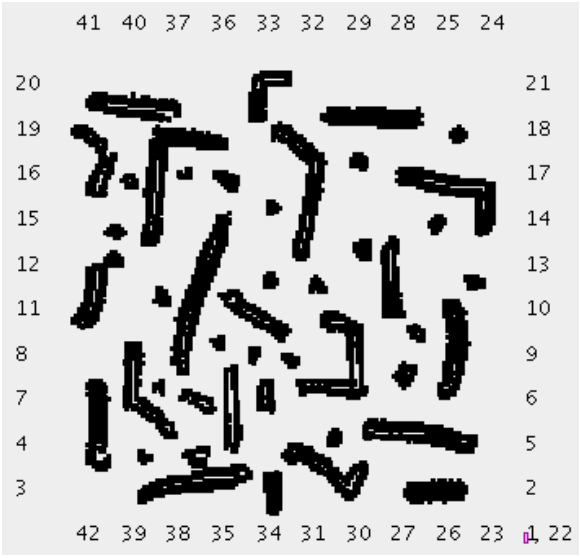


Figure 52: The sequence of goals used in the quantitative tests.

B.2.1 Experimental Systems vs. Baseline

In this experiment, a system using a planning cost model learned from the training experiences was compared to a baseline system. This experiment was run in the environment shown in Figure 51(b). The robot was tasked with executing the goals shown in Figure 52 in order while the robot's performance was measured. A comparison of the performance of the systems is shown in Table 11. The experimental system finished the course in 17% less time than the baseline system.

Table 11: Results of timed test of baseline and experimental systems. The experimental system improved performance over the baseline by 17%, with a statistically significant p-value of .02.

	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	7745 sec	5124 m	N/A	N/A
Experimental System (KNN $k = 5$)	6434 sec	5577 m	17%	.02

The raw data used to compile these results is shown in Tables 12 and 13. Table 12 contains the data collected for the baseline system. Each goal ID is enumerated (matching the numbers in Figure 52), along with the goal's x and y-coordinates and the goals straight-line distance from the last goal. Also listed is the time in the mission each goal is achieved, and the time elapsed since achieving the previous goal. Table 13 contains the same data for the experimental system.

Table 12: Baseline performance over the set of goals shown in Figure 52.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Start	0	0	0	26159	0
Goal 0	160	130	206	242400	216.241
Goal 1	0	0	206	1015499	773.099
Goal 2	15	0	15	1060559	45.06
Goal 3	15	150	150	1294559	234.0
Goal 4	30	150	15	1322279	27.72
Goal 5	30	0	150	1616159	293.88
Goal 6	45	0	15	1642079	25.92
Goal 7	45	150	150	2069399	427.32
Goal 8	60	150	15	2149199	79.8
Goal 9	60	0	150	2544779	395.58
Goal 10	75	0	15	2575199	30.42
Goal 11	75	150	150	3042479	467.28
Goal 12	90	150	15	3089699	47.22
Goal 13	90	0	150	3507779	418.08
Goal 14	105	0	15	3563999	56.22
Goal 15	105	150	150	4049759	485.76
Goal 16	120	150	15	4073999	24.24
Goal 17	120	0	150	4367160	293.161
Goal 18	135	0	15	4443660	76.5
Goal 19	135	150	150	4816860	373.2
Continued on next page.					

Table 12 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 20	150	150	15	4820880	4.02
Goal 21	150	0	150	5042760	221.88
Goal 22	0	0	150	5145180	102.42
Goal 23	0	15	15	5206800	61.62
Goal 24	150	15	150	5324280	117.48
Goal 25	150	30	15	5379060	54.78
Goal 26	0	30	150	5607660	228.6
Goal 27	0	45	15	5628000	20.34
Goal 28	150	45	150	5960880	332.88
Goal 29	150	60	15	5997360	36.48
Goal 30	0	60	150	6361500	364.14
Goal 31	0	75	15	6417900	56.4
Goal 32	150	75	150	6746400	328.5
Goal 33	150	90	15	6974760	228.36
Goal 34	0	90	150	7331580	356.82
Goal 35	0	105	15	7396080	64.5
Goal 36	150	105	150	7908780	512.7
Goal 37	150	120	15	7977360	68.58
Goal 38	0	120	150	8244480	267.12
Goal 39	0	135	15	8248800	4.32
Goal 40	150	135	150	8542860	294.06
Goal 41	150	150	15	8548980	6.12
Continued on next page.					

Table 12 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 42	0	150	150	8794680	245.7

Table 13: Experimental system performance over the set of goals shown in Figure 52.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Start	0	0	0	618599	0
Goal 0	160	130	206	766499	147.9
Goal 1	0	0	206	1109879	343.38
Goal 2	15	0	15	1224059	114.18
Goal 3	15	150	150	1410779	186.72
Goal 4	30	150	15	1443479	32.7
Goal 5	30	0	150	1704059	260.58
Goal 6	45	0	15	1715699	11.64
Goal 7	45	150	150	1960079	244.38
Goal 8	60	150	15	1978379	18.3
Goal 9	60	0	150	2263499	285.12
Goal 10	75	0	15	2272499	9.0
Goal 11	75	150	150	2681759	409.26
Goal 12	90	150	15	2719679	37.92
Continued on next page.					

Table 13 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 13	90	0	150	3180779	461.1
Goal 14	105	0	15	3235259	54.48
Goal 15	105	150	150	3540359	305.1
Goal 16	120	150	15	3708719	168.36
Goal 17	120	0	150	3951059	242.34
Goal 18	135	0	15	4048979	97.92
Goal 19	135	150	150	4276860	227.881
Goal 20	150	150	15	4401000	124.14
Goal 21	150	0	150	4552380	151.38
Goal 22	0	0	150	4644360	91.98
Goal 23	0	15	15	4693560	49.2
Goal 24	150	15	150	4880460	186.9
Goal 25	150	30	15	4939380	58.92
Goal 26	0	30	150	5262360	322.98
Goal 27	0	45	15	5274480	12.12
Goal 28	150	45	150	5594280	319.8
Goal 29	150	60	15	5609880	15.6
Goal 30	0	60	150	5943000	333.12
Goal 31	0	75	15	5968560	25.56
Goal 32	150	75	150	6456000	487.44
Goal 33	150	90	15	6514500	58.5
Goal 34	0	90	150	6820500	306.0
Continued on next page.					

Table 13 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 35	0	105	15	6883080	62.58
Goal 36	150	105	150	7395780	512.7
Goal 37	150	120	15	7441500	45.72
Goal 38	0	120	150	7747200	305.7
Goal 39	0	135	15	7833660	86.46
Goal 40	150	135	150	8058660	225
Goal 41	150	150	15	8192160	133.5
Goal 42	0	150	150	8269980	77.82

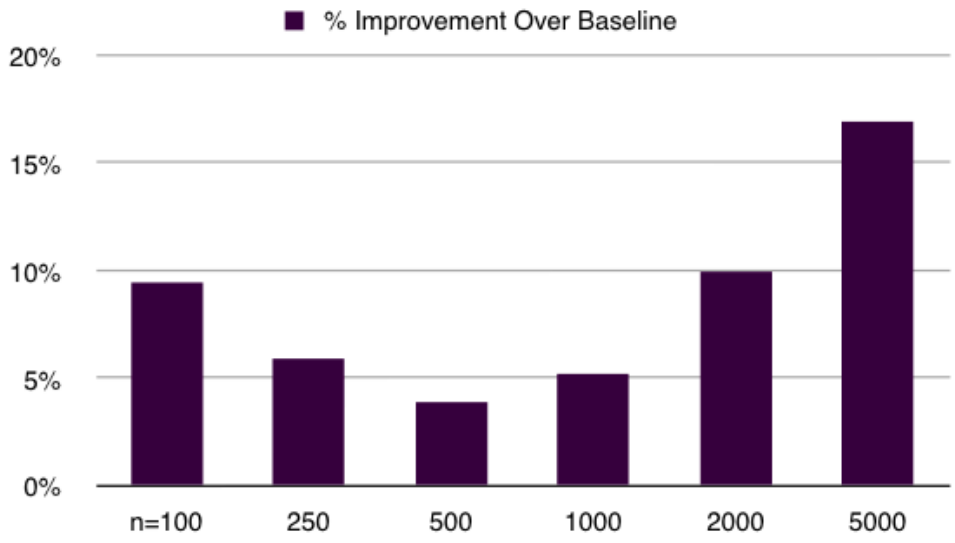
B.2.2 Experimental System vs. Number of Learning Experiences

In this experiment, systems using a planning cost model learned from varying numbers of training experiences were compared. This experiment was run in the environment shown in Figure 51(b). In each run, the robot was tasked with executing the goals shown in Figure 52 in order while the robot's performance was measured. A comparison of the performance of the classifiers in a 10-fold cross-validation is shown in Table 14. In system experimentation, as the number of training experiences used increases, the performance improves. This trend is shown in Figure 53.

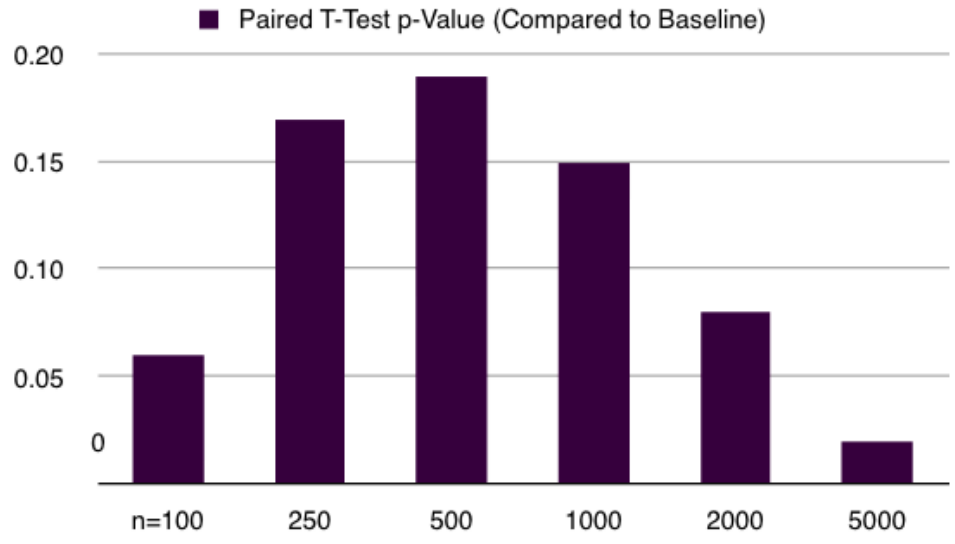
Table 14: Results of a 10-fold cross validation test on several k-nearest neighbor instantiations, using varying numbers of examples to train. The value of k was set to $k = 5$ throughout the tests.

	$n = 100$	$n = 250$	$n = 500$	$n = 1000$	$n = 2500$	$n = 5000$
Average Relative Error	112%	95%	87%	74%	71%	64%
Correlation	.33	.43	.49	.54	.56	.65

The raw data used to compile these results is shown in Tables 15, 16, 17, 18 and 19. Each goal ID is enumerated (matching the numbers in Figure 52), along with the goal's x and y-coordinates and the goals straight-line distance from the last goal. Also listed is the time in the mission each goal is achieved, and the time elapsed since achieving the previous goal.



(a)



(b)

Figure 53: System performance improvement over the baseline planner, as a function of the number of examples used in training a k-nearest neighbor regression model. (a) shows the improvement over the baseline system, while (b) shows the paired t-test p-value for each instantiation.

Table 15: Experimental system performance over the set of goals shown in Figure 52 for $N = 2500$.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Start	0	0	0	618599	0
Goal 0	160	130	206	766499	147.9
Goal 1	0	0	206	1109879	343.38
Goal 2	15	0	15	1224059	114.18
Goal 3	15	150	150	1410779	186.72
Goal 4	30	150	15	1443479	32.7
Goal 5	30	0	150	1704059	260.58
Goal 6	45	0	15	1715699	11.64
Goal 7	45	150	150	1960079	244.38
Goal 8	60	150	15	1978379	18.3
Goal 9	60	0	150	2263499	285.12
Goal 10	75	0	15	2272499	9.0
Goal 11	75	150	150	2681759	409.26
Goal 12	90	150	15	2719679	37.92
Goal 13	90	0	150	3180779	461.1
Goal 14	105	0	15	3235259	54.48
Goal 15	105	150	150	3540359	305.1
Goal 16	120	150	15	3708719	168.36
Goal 17	120	0	150	3951059	242.34
Goal 18	135	0	15	4048979	97.92
Continued on next page.					

Table 15 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 19	135	150	150	4276860	227.881
Goal 20	150	150	15	4401000	124.14
Goal 21	150	0	150	4552380	151.38
Goal 22	0	0	150	4644360	91.98
Goal 23	0	15	15	4693560	49.2
Goal 24	150	15	150	4880460	186.9
Goal 25	150	30	15	4939380	58.92
Goal 26	0	30	150	5262360	322.98
Goal 27	0	45	15	5274480	12.12
Goal 28	150	45	150	5594280	319.8
Goal 29	150	60	15	5609880	15.6
Goal 30	0	60	150	5943000	333.12
Goal 31	0	75	15	5968560	25.56
Goal 32	150	75	150	6456000	487.44
Goal 33	150	90	15	6514500	58.5
Goal 34	0	90	150	6820500	306.0
Goal 35	0	105	15	6883080	62.58
Goal 36	150	105	150	7395780	512.7
Goal 37	150	120	15	7441500	45.72
Goal 38	0	120	150	7747200	305.7
Goal 39	0	135	15	7833660	86.46
Goal 40	150	135	150	8058660	225
Continued on next page.					

Table 15 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 41	150	150	15	8192160	133.5
Goal 42	0	150	150	8269980	77.82

Table 16: Experimental system performance over the set of goals shown in Figure 52 for $N = 1000$.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Start	0	0	0	195000	0
Goal 0	160	130	206	354059	159.059
Goal 1	0	0	206	656459	302.4
Goal 2	15	0	15	750959	94.5
Goal 3	15	150	150	981299	230.34
Goal 4	30	150	15	1014599	33.3
Goal 5	30	0	150	1272359	257.76
Goal 6	45	0	15	1283699	11.34
Goal 7	45	150	150	1534199	250.5
Goal 8	60	150	15	1554179	19.98
Goal 9	60	0	150	2058479	504.3
Goal 10	75	0	15	2069279	10.8
Goal 11	75	150	150	2509199	439.92
Continued on next page.					

Table 16 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 12	90	150	15	2552279	43.08
Goal 13	90	0	150	2980859	428.58
Goal 14	105	0	15	2981999	1.14
Goal 15	105	150	150	3575759	593.76
Goal 16	120	150	15	3669059	93.3
Goal 17	120	0	150	3846899	177.84
Goal 18	135	0	15	3975599	128.7
Goal 19	135	150	150	4233299	257.7
Goal 20	150	150	15	4236359	3.06
Goal 21	150	0	150	4447860	211.501
Goal 22	0	0	150	4577460	129.6
Goal 23	0	15	15	4624980	47.52
Goal 24	150	15	150	4870560	245.58
Goal 25	150	30	15	4928880	58.32
Goal 26	0	30	150	5226300	297.42
Goal 27	0	45	15	5238180	11.88
Goal 28	150	45	150	5636760	398.58
Goal 29	150	60	15	5744100	107.34
Goal 30	0	60	150	6135300	391.2
Goal 31	0	75	15	6180600	45.3
Goal 32	150	75	150	6538560	357.96
Goal 33	150	90	15	6609480	70.92
Continued on next page.					

Table 16 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 34	0	90	150	6959400	349.92
Goal 35	0	105	15	7050900	91.5
Goal 36	150	105	150	7412160	361.26
Goal 37	150	120	15	7480680	68.52
Goal 38	0	120	150	7647960	167.28
Goal 39	0	135	15	7772400	124.44
Goal 40	150	135	150	8001480	229.08
Goal 41	150	150	15	8204400	202.92
Goal 42	0	150	150	8301060	96.66

Table 17: Experimental system performance over the set of goals shown in Figure 52 for $N = 500$.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Start	0	0	0	183960	0
Goal 0	160	130	206	358079	174.119
Goal 1	0	0	206	610199	252.12
Goal 2	15	0	15	611279	1.08
Goal 3	15	150	150	755159	143.88
Goal 4	30	150	15	782999	27.84
Continued on next page.					

Table 17 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 5	30	0	150	1041899	258.9
Goal 6	45	0	15	1064699	22.8
Goal 7	45	150	150	1293899	229.2
Goal 8	60	150	15	1301999	8.1
Goal 9	60	0	150	1739459	437.46
Goal 10	75	0	15	1750979	11.52
Goal 11	75	150	150	2168099	417.12
Goal 12	90	150	15	2208059	39.96
Goal 13	90	0	150	2660699	452.64
Goal 14	105	0	15	2707199	46.5
Goal 15	105	150	150	3177299	470.1
Goal 16	120	150	15	3260399	83.1
Goal 17	120	0	150	3427079	166.68
Goal 18	135	0	15	3566279	139.2
Goal 19	135	150	150	3771899	205.62
Goal 20	150	150	15	3773579	1.68
Goal 21	150	0	150	3980999	207.42
Goal 22	0	0	150	4105079	124.08
Goal 23	0	15	15	4145579	40.5
Goal 24	150	15	150	4396560	250.981
Goal 25	150	30	15	4458000	61.44
Goal 26	0	30	150	4697160	239.16
Continued on next page.					

Table 17 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 27	0	45	15	4721280	24.12
Goal 28	150	45	150	5150400	429.12
Goal 29	150	60	15	5368980	218.58
Goal 30	0	60	150	5709000	340.02
Goal 31	0	75	15	5760000	51
Goal 32	150	75	150	6078300	318.3
Goal 33	150	90	15	6148800	70.5
Goal 34	0	90	150	6495780	346.98
Goal 35	0	105	15	6523800	28.02
Goal 36	150	105	150	7273680	749.88
Goal 37	150	120	15	7329180	55.5
Goal 38	0	120	150	7497900	168.72
Goal 39	0	135	15	7627380	129.48
Goal 40	150	135	150	7831260	203.88
Goal 41	150	150	15	7834380	3.12
Goal 42	0	150	150	8036160	201.78

Table 18: Experimental system performance over the set of goals shown in Figure 52 for $N = 250$.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Start	0	0	0	618599	0
Goal 0	160	130	206	766499	147.9
Goal 1	0	0	206	1109879	343.38
Goal 2	15	0	15	1224059	114.18
Goal 3	15	150	150	1410779	186.72
Goal 4	30	150	15	1443479	32.7
Goal 5	30	0	150	1704059	260.58
Goal 6	45	0	15	1715699	11.64
Goal 7	45	150	150	1960079	244.38
Goal 8	60	150	15	1978379	18.3
Goal 9	60	0	150	2263499	285.12
Goal 10	75	0	15	2272499	9.0
Goal 11	75	150	150	2681759	409.26
Goal 12	90	150	15	2719679	37.92
Goal 13	90	0	150	3180779	461.1
Goal 14	105	0	15	3235259	54.48
Goal 15	105	150	150	3540359	305.1
Goal 16	120	150	15	3708719	168.36
Goal 17	120	0	150	3951059	242.34
Goal 18	135	0	15	4048979	97.92
Continued on next page.					

Table 18 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 19	135	150	150	4276860	227.881
Goal 20	150	150	15	4401000	124.14
Goal 21	150	0	150	4552380	151.38
Goal 22	0	0	150	4644360	91.98
Goal 23	0	15	15	4693560	49.2
Goal 24	150	15	150	4880460	186.9
Goal 25	150	30	15	4939380	58.92
Goal 26	0	30	150	5262360	322.98
Goal 27	0	45	15	5274480	12.12
Goal 28	150	45	150	5594280	319.8
Goal 29	150	60	15	5609880	15.6
Goal 30	0	60	150	5943000	333.12
Goal 31	0	75	15	5968560	25.56
Goal 32	150	75	150	6456000	487.44
Goal 33	150	90	15	6514500	58.5
Goal 34	0	90	150	6820500	306.0
Goal 35	0	105	15	6883080	62.58
Goal 36	150	105	150	7395780	512.7
Goal 37	150	120	15	7441500	45.72
Goal 38	0	120	150	7747200	305.7
Goal 39	0	135	15	7833660	86.46
Goal 40	150	135	150	8058660	225
Continued on next page.					

Table 18 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 41	150	150	15	8192160	133.5
Goal 42	0	150	150	8269980	77.82

Table 19: Experimental system performance over the set of goals shown in Figure 52 for $N = 100$.

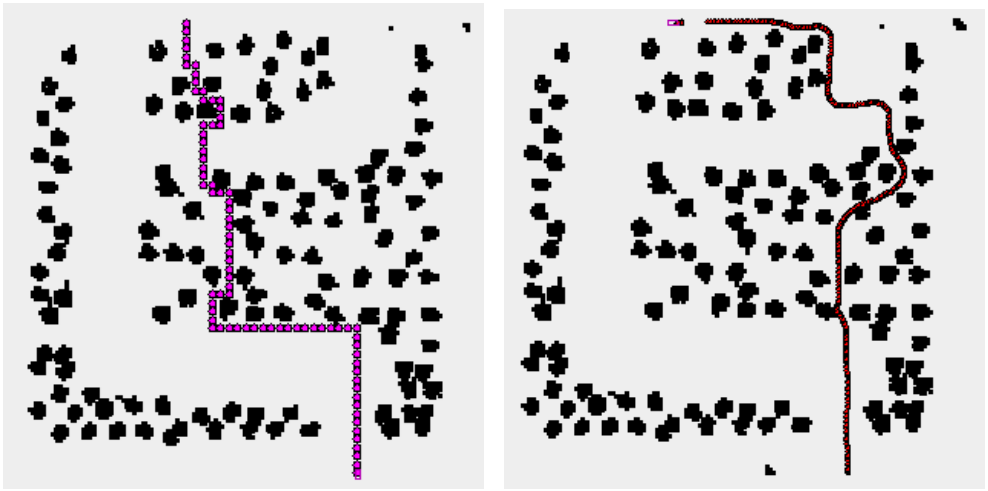
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Start	0	0	0	528599	0
Goal 0	160	130	206	680759	152.16
Goal 1	0	0	206	940379	259.62
Goal 2	15	0	15	1053899	113.52
Goal 3	15	150	150	1230899	177
Goal 4	30	150	15	1263479	32.58
Goal 5	30	0	150	1524059	260.58
Goal 6	45	0	15	1532999	8.94
Goal 7	45	150	150	1764299	231.3
Goal 8	60	150	15	1778579	14.28
Goal 9	60	0	150	2300279	521.7
Goal 10	75	0	15	2312699	12.42
Goal 11	75	150	150	2718779	406.08
Continued on next page.					

Table 19 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 12	90	150	15	2760479	41.7
Goal 13	90	0	150	3181979	421.5
Goal 14	105	0	15	3240299	58.32
Goal 15	105	150	150	3699899	459.6
Goal 16	120	150	15	3748859	48.96
Goal 17	120	0	150	3904979	156.12
Goal 18	135	0	15	4098899	193.92
Goal 19	135	150	150	4338300	239.401
Goal 20	150	150	15	4341900	3.6
Goal 21	150	0	150	4544460	202.56
Goal 22	0	0	150	4676880	132.42
Goal 23	0	15	15	4733700	56.82
Goal 24	150	15	150	4937460	203.76
Goal 25	150	30	15	5002560	65.1
Goal 26	0	30	150	5257680	255.12
Goal 27	0	45	15	5286960	29.28
Goal 28	150	45	150	5657400	370.44
Goal 29	150	60	15	5682300	24.9
Goal 30	0	60	150	6112800	430.5
Goal 31	0	75	15	6158100	45.3
Goal 32	150	75	150	6515580	357.48
Goal 33	150	90	15	6586560	70.98
Continued on next page.					

Table 19 – continued from previous page					
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Time Achieved (ms)	Elapsed Time (s)
Goal 34	0	90	150	6963300	376.74
Goal 35	0	105	15	7024740	61.44
Goal 36	150	105	150	7453260	428.52
Goal 37	150	120	15	7502880	49.62
Goal 38	0	120	150	7681800	178.92
Goal 39	0	135	15	7890060	208.26
Goal 40	150	135	150	8113260	223.2
Goal 41	150	150	15	8272560	159.3
Goal 42	0	150	150	8399760	127.2

B.2.3 Qualitative Performance

In this experiment, the system trained on the full set of training experiences is compared qualitatively to the baseline system. This experiment was run in the environment shown in Figure 51(c). Figure 54 shows the baseline system's proposed plan (at the beginning of the run) and executed trajectory. The baseline system plans a path "through the woods" which results in a twisting, slow trajectory. Figure 55 shows the experimental system's proposed plan (again, at the beginning of the run) and executed trajectory. The experimental system plans a slightly longer path, but executes the plan in less time. Table 20 contains quantitative data comparing the two respective runs.

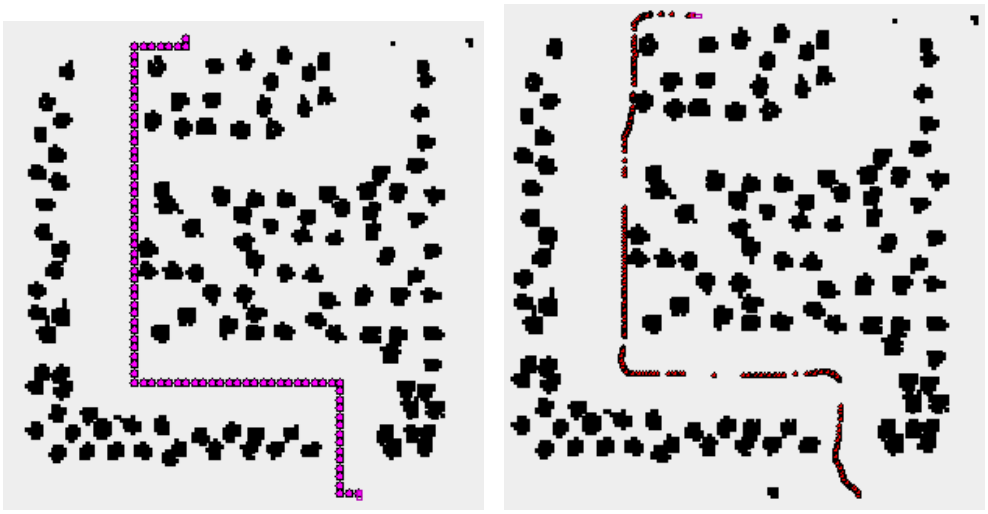


(a) (b)

Figure 54: In (a), the planned path from the robot to the goal through the "path in the woods" environment using the baseline planner. The robot is at the bottom of the image. The goal is at the top of the image. The planned path is shown by pink waypoints. Obstacles in the map are shown in black. In (b), the actual path taken by the system (trajectory in red).

B.2.4 Experimental System Performance vs. Cost Function

In this experiment, the experimental system is trained on a large set of training data based on three different cost functions, h_{c1}, h_{c2} and h_{c3} (see Equation 21). This



(a) (b)

Figure 55: In (a), the planned path from the robot to the goal through the "path in the woods" environment, using the experimental planner. The resulting plan is longer given a constant velocity model of the robot, but when used as input to the reactive layer, as shown in (b), reduces mission time by 25% over the plan shown in Figure 54(a).

Table 20: Quantitative comparison of system performance on the "path in the woods" course.

System	Start Time (ms)	End Time (ms)	Elapsed Time (s)
Baseline	21179	37659	351.48
Experimental	1718759	1984859	266.1

experiment was run in the environment shown in Figure 51(b). The robot was tasked with executing the goals shown in Figure 52 in order while the robot's performance was measured. A comparison of the performance of the systems is shown in Tables 21 (cost function h_{c1}), 22 (cost function h_{c2}) and 23 (cost function h_{c3}.

The raw data used to compile these results is shown in Tables 24, 25, 26, 27, 28 and 29. Table 24 contains the data collected for the baseline system, as measured by fuel model h_{c1}. Each goal ID is enumerated (matching the numbers in Figure 52), along with the goal's x and y-coordinates and the goals' straight-line distance from the last goal. Also listed is the fuel used during each segment of the course. Table

25 contains the same data for the experimental system based on cost function h_{c1}. Tables 26 and 27 contains the same data for the baseline and experimental system based on cost function h_{c2}. Tables 28 and 29 contains the same data for the baseline and experimental system based on cost function h_{c3}.

Table 21: Results of the experiment comparing the performance of the baseline system and the experimental system, using cost measurement function 1, h_{c1}. The experimental system improved performance over the baseline by nearly 13%, with a statistically significant p-value of .04.

	Simulated Fuel to Complete Test	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	585,606.86 units	6437 sec	4026 m	N/A	N/A
Experimental System (KNN $k = 5$)	511,158.26 units	5879 sec	3922 m	12.7%	.04

Table 22: Results of the experiment comparing the performance of the baseline system and the experimental system, using cost measurement function 1, h_{c2}. The experimental system improved performance over the baseline by 30%, with a (very) statistically significant p-value of . 000002.

	Simulated Fuel to Complete Test	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	248,187.99 units	6437 sec	4026 m	N/A	N/A
Experimental System (KNN $k = 5$)	172,472.05 units	5657 sec	4015 m	30.5%	.000002

Table 23: Results of the experiment comparing the performance of the baseline system and the experimental system, using cost measurement function 1, h_{c1}. The experimental system performed worse than the baseline by 17%. This result, however is not surprising, as the learned planning function does not have control over the primary component of the cost function, velocity.

	Simulated Fuel to Complete Test	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	1,846,857.7 units	6437 sec	4026 m	N/A	N/A
Experimental System (KNN $k = 15$)	1,582,452.9 units	5867 sec	4183 m	-16.7%	.04

Table 24: Baseline system performance over the set of goals shown in Figure 52, evaluated based on fuel model h_{c1}.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 1 (Start)	0	0	0	0
Goal 2	15	0	15	3379.6
Goal 3	15	150	150	15991.29
Goal 4	30	150	15	636.7
Goal 5	30	0	150	24507.34
Goal 6	45	0	15	1111.77
Goal 7	45	150	150	52266.19
Goal 8	60	150	15	1807.32
Goal 9	60	0	150	31333.14
Continued on next page.				

Table 24 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 10	75	0	15	123.2
Goal 11	75	150	150	52770.64
Goal 12	90	150	15	810.76
Goal 13	90	0	150	40391.22
Goal 14	105	0	15	5165.76
Goal 15	105	150	150	44754.78
Goal 16	120	150	15	541.1
Goal 17	120	0	150	30920
Goal 18	135	0	15	2301.16
Goal 19	135	150	150	25493.74
Goal 20	150	150	15	9409.15
Goal 21	150	0	150	15309.01
Goal 22	0	0	150	6870.51
Goal 23	0	15	15	10.3
Goal 24	150	15	150	11066.34
Goal 25	150	30	15	6919.65
Goal 26	0	30	150	20357.29
Goal 27	0	45	15	2009.7
Goal 28	150	45	150	34570.14
Goal 29	150	60	15	36.78
Goal 30	0	60	150	35092.98
Goal 31	0	75	15	3832.66
Continued on next page.				

Table 24 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 32	150	75	150	24457.75
Goal 33	150	90	15	373.5
Goal 34	0	90	150	48962.84
Goal 35	0	105	15	41.34
Goal 36	150	105	150	49179.95
Goal 37	150	120	15	7422.07
Goal 38	0	120	150	28182.22
Goal 39	0	135	15	58237.99
Goal 40	150	135	150	203.2
Goal 41	150	150	15	14543.3
Goal 42	0	150	150	12427.2

Table 25: Experimental system performance over the set of goals shown in Figure 52, evaluated based on fuel model h_{c1}.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 1 (Start)	0	0	0	0.0
Goal 2	15	0	15	34.67
Goal 3	15	150	150	22727.57
Continued on next page.				

Table 25 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 4	30	150	15	242.67
Goal 5	30	0	150	21738.82
Goal 6	45	0	15	3394.26
Goal 7	45	150	150	35426.29
Goal 8	60	150	15	858.74
Goal 9	60	0	150	30229.57
Goal 10	75	0	15	6275.65
Goal 11	75	150	150	32531.83
Goal 12	90	150	15	45.67
Goal 13	90	0	150	36958.02
Goal 14	105	0	15	2630.03
Goal 15	105	150	150	39286.77
Goal 16	120	150	15	102.59
Goal 17	120	0	150	26447.23
Goal 18	135	0	15	4416.78
Goal 19	135	150	150	14393.82
Goal 20	150	150	15	717.52
Goal 21	150	0	150	28090.56
Goal 22	0	0	150	10097.29
Goal 23	0	15	15	117.8
Goal 24	150	15	150	11273.29
Goal 25	150	30	15	65.4
Continued on next page.				

Table 25 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 26	0	30	150	21321.88
Goal 27	0	45	15	32.1
Goal 28	150	45	150	26116.04
Goal 29	150	60	15	650.59
Goal 30	0	60	150	30864.28
Goal 31	0	75	15	600.38
Goal 32	150	75	150	27814.36
Goal 33	150	90	15	501.86
Goal 34	0	90	150	44378.16
Goal 35	0	105	15	42.1
Goal 36	150	105	150	38560.05
Goal 37	150	120	15	1305.21
Goal 38	0	120	150	22999.72
Goal 39	0	135	15	986.79
Goal 40	150	135	150	20064.74
Goal 41	150	150	15	78.8
Goal 42	0	150	150	13796.4

Table 26: Baseline system performance over the set of goals shown in Figure 52, evaluated based on fuel model h_{c2}.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 1 (Start)	0	0	0	0
Goal 2	15	0	15	170.32
Goal 3	15	150	150	6559.61
Goal 4	30	150	15	60.28
Goal 5	30	0	150	12069.47
Goal 6	45	0	15	151.87
Goal 7	45	150	150	14873.1
Goal 8	60	150	15	134.8
Goal 9	60	0	150	14681.27
Goal 10	75	0	15	34.89
Goal 11	75	150	150	16843.04
Goal 12	90	150	15	71.75
Goal 13	90	0	150	16281.29
Goal 14	105	0	15	346.17
Goal 15	105	150	150	18829.17
Goal 16	120	150	15	24.5
Goal 17	120	0	150	13648.77
Goal 18	135	0	15	696.84
Goal 19	135	150	150	15049.19
Continued on next page.				

Table 26 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 20	150	150	15	735.86
Goal 21	150	0	150	9920.95
Goal 22	0	0	150	5721.19
Goal 23	0	15	15	109.87
Goal 24	150	15	150	6847.39
Goal 25	150	30	15	1718.63
Goal 26	0	30	150	9786.22
Goal 27	0	45	15	775.53
Goal 28	150	45	150	13749.53
Goal 29	150	60	15	67.9
Goal 30	0	60	150	10853.18
Goal 31	0	75	15	769.09
Goal 32	150	75	150	16050.09
Goal 33	150	90	15	50.27
Goal 34	0	90	150	17104.65
Goal 35	0	105	15	56.7
Goal 36	150	105	150	15644.81
Goal 37	150	120	15	2623.52
Goal 38	0	120	150	19396.26
Goal 39	0	135	15	29180.66
Goal 40	150	135	150	34109.6
Goal 41	150	150	15	89.3
Continued on next page.				

Table 26 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 42	0	150	150	8658.4

Table 27: Experimental system performance over the set of goals shown in Figure 52, evaluated based on fuel model h_{c2}.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 1 (Start)	0	0	0	0.0
Goal 2	15	0	15	43.6
Goal 3	15	150	150	6640.41
Goal 4	30	150	15	120.62
Goal 5	30	0	150	7526.4
Goal 6	45	0	15	376.35
Goal 7	45	150	150	10513.42
Goal 8	60	150	15	848.68
Goal 9	60	0	150	12984.26
Goal 10	75	0	15	684.37
Goal 11	75	150	150	11195.49
Goal 12	90	150	15	87.9
Goal 13	90	0	150	12054.23
Continued on next page.				

Table 27 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 14	105	0	15	466.18
Goal 15	105	150	150	11426.84
Goal 16	120	150	15	192.61
Goal 17	120	0	150	13422.81
Goal 18	135	0	15	82.75
Goal 19	135	150	150	10136.43
Goal 20	150	150	15	89.7
Goal 21	150	0	150	5857.63
Goal 22	0	0	150	13585.26
Goal 23	0	15	15	76.9
Goal 24	150	15	150	5736.12
Goal 25	150	30	15	14.31
Goal 26	0	30	150	9036.21
Goal 27	0	45	15	390.06
Goal 28	150	45	150	9190.71
Goal 29	150	60	15	2667.75
Goal 30	0	60	150	7254.92
Goal 31	0	75	15	1.77
Goal 32	150	75	150	8710.02
Goal 33	150	90	15	125.09
Goal 34	0	90	150	10575.37
Goal 35	0	105	15	78.2
Continued on next page.				

Table 27 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 36	150	105	150	7521.73
Goal 37	150	120	15	163.33
Goal 38	0	120	150	12689.05
Goal 39	0	135	15	98.2
Goal 40	150	135	150	10157.36
Goal 41	150	150	15	67.2
Goal 42	0	150	150	9873.2

Table 28: Baseline system performance over the set of goals shown in Figure 52, evaluated based on fuel model h_{c3}.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 1 (Start)	0	0	0	0
Goal 2	15	0	15	59.26
Goal 3	15	150	150	140513.88
Goal 4	30	150	15	22.91
Goal 5	30	0	150	118415.79
Goal 6	45	0	15	48.82
Goal 7	45	150	150	38296.65
Continued on next page.				

Table 28 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 8	60	150	15	34.38
Goal 9	60	0	150	29684.31
Goal 10	75	0	15	45.78
Goal 11	75	150	150	53953.69
Goal 12	90	150	15	21.61
Goal 13	90	0	150	61394.16
Goal 14	105	0	15	8215.45
Goal 15	105	150	150	41520.82
Goal 16	120	150	15	97.5
Goal 17	120	0	150	110771.48
Goal 18	135	0	15	1485.47
Goal 19	135	150	150	70247.22
Goal 20	150	150	15	174.74
Goal 21	150	0	150	151853.53
Goal 22	0	0	150	209555.56
Goal 23	0	15	15	92.87
Goal 24	150	15	150	170526.84
Goal 25	150	30	15	4039.28
Goal 26	0	30	150	211089.76
Goal 27	0	45	15	3475.14
Goal 28	150	45	150	50371.44
Goal 29	150	60	15	76.45
Continued on next page.				

Table 28 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 30	0	60	150	40639.26
Goal 31	0	75	15	1595.88
Goal 32	150	75	150	31268.55
Goal 33	150	90	15	108.52
Goal 34	0	90	150	70196.48
Goal 35	0	105	15	43.56
Goal 36	150	105	150	55414.01
Goal 37	150	120	15	4196.58
Goal 38	0	120	150	136295.00
Goal 39	0	135	15	230289.92
Goal 40	150	135	150	78.1
Goal 41	150	150	15	109381.91
Goal 42	0	150	150	34.51

Table 29: Experimental system performance over the set of goals shown in Figure 52, evaluated based on fuel model h_{c3}.

Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 1 (Start)	0	0	0	0.0
Continued on next page.				

Table 29 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 2	15	0	15	33.45
Goal 3	15	150	150	129273.97
Goal 4	30	150	15	24.3
Goal 5	30	0	150	214068.64
Goal 6	45	0	15	295.09
Goal 7	45	150	150	31849.61
Goal 8	60	150	15	908.4
Goal 9	60	0	150	91300.13
Goal 10	75	0	15	452.64
Goal 11	75	150	150	47983.73
Goal 12	90	150	15	12.07
Goal 13	90	0	150	24722.15
Goal 14	105	0	15	97.5
Goal 15	105	150	150	38651.23
Goal 16	120	150	15	105.4
Goal 17	120	0	150	199619.52
Goal 18	135	0	15	109.8
Goal 19	135	150	150	147637.36
Goal 20	150	150	15	134.2
Goal 21	150	0	150	147658.36
Goal 22	0	0	150	117778.8
Goal 23	0	15	15	113.4
Continued on next page.				

Table 29 – continued from previous page				
Goal ID	Goal x (m)	Goal y (m)	Distance to Goal (m)	Fuel Units Used (per segment)
Goal 24	150	15	150	165915.08
Goal 25	150	30	15	16.23
Goal 26	0	30	150	151852.44
Goal 27	0	45	15	104.91
Goal 28	150	45	150	30123.81
Goal 29	150	60	15	63.43
Goal 30	0	60	150	69100.17
Goal 31	0	75	15	98.9
Goal 32	150	75	150	40615.9
Goal 33	150	90	15	42.65
Goal 34	0	90	150	35987.92
Goal 35	0	105	15	167.52
Goal 36	150	105	150	79272.12
Goal 37	150	120	15	78.44
Goal 38	0	120	150	201225.56
Goal 39	0	135	15	65.77
Goal 40	150	135	150	149378.11
Goal 41	150	150	15	46.94
Goal 42	0	150	150	180981.81

REFERENCES

[1] Albus, J. S., "4d/rcs: a reference model architecture for intelligent unmanned ground vehicles," in *In Proceedings of SPIE Aerosense Conference*, pp. 1–5, 2002.

[2] Alur, R., Belta, C., Ivančić, F., Kumar, V., Rubin, H., Schug, J., Sokolsky, O., and Webb, J., "Visual programming for modeling and simulation of biomolecular regulatory networks," *Lecture Notes in Computer Science*, vol. 2552, pp. 702–713, 2002.

[3] Arkin, R., "Motor schema based mobile robot navigation," *International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.

[4] Arkin, R. C., *Behavior-Based Robotics*. The MIT Press, 1998.

[5] Arkin, R. C. and Balch, T., "Aura: Principles and practice in review," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, pp. 175–189, 1997.

[6] Balch, T., *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, December 1998.

[7] Balch, T. and Arkin, R. C., "Motor schema-based formation control for multiagent robot teams," in *In Proceedings of the First International Conference on Multi-Agent Systems*, pp. 10–16, AAAI Press, 1995.

[8] Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and Pappas, G., "Symbolic planning and control of robot motion: State of the art and grand challenges," *IEEE Robotics and Automation Magazine*, vol. 14, March 2007.

[9] Branicky, M. S., Borkar, V. S., and Mitter, S. K., "A unified framework for hybrid control: Model and optimal control theory," *IEEE Transactions on Automatic Control*, vol. 43, pp. 31–45, 1998.

[10] Brock, O. and Khatib, O., "High-speed navigation using the global dynamic window approach," in *In IEEE Int. Conf. on Robotics and Automation*, pp. 341–346, 1999.

[11] Brooks, R., "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, p. 14, 1986.

[12] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S., *Principles of Robot Motion: Theory, Algorithms, and Implementation*. The MIT Press, 2005.

[13] DARPA, "Learning applied to ground robots (lagr) proposer information pamphlet." BAA-04-25, 2004.

[14] DARPA, "Darpa urban challenge rules and evaluation criteria." www.darpa.mil/grandchallenge/rules.asp, 2008.

[15] Egerstedt, M., Azuma, S.-I., and Wardi, Y., "Optimal timing control of switched linear systems based on partial information," *Nonlinear Analysis: Theory, Methods and Applications*, vol. 65, pp. 1736–1750, November 2006.

[16] Egerstedt, M., Wardi, Y., and Axelsson, H., "Transition-time optimization for switched systems," *IEEE Transactions on Automatic Control*, vol. 51, pp. 110–115, January 2006.

[17] Fox, D., Burgard, W., and Thrun, S., "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, 1997.

[18] Fox, D., Burgard, W., and Thrun, S., "A hybrid collision avoidance method for mobile robots," in *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998.

[19] Frank, E., Hall, M. A., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I. H., and Trigg, L., "Weka - a machine learning workbench for data mining," in *The Data Mining and Knowledge Discovery Handbook* (Maimon, O. and Rokach, L., eds.), pp. 1305–1314, Springer, 2005.

[20] Frehse, G., Krogh, B., and Rutenbar, R., "Time domain verification of oscillator circuit properties," *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 3, pp. 9–22, 2006.

[21] Gat, E., "Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots," *SIGART Bulletin*, vol. 2, no. 4, pp. 70–74, 1991.

[22] Gat, E., Bonnasso, R. P., Murphy, R., and Press, A., "On three-layer architectures," in *Artificial Intelligence and Mobile Robots*, pp. 195–210, AAAI Press, 1998.

[23] Gat, E. and Dorais, G., "Robot navigation by conditional sequencing," in *IEEE International Conference on Robotics and Automation*, pp. 1293–99, 1994.

[24] Gerkey, B. P., Vaughan, R. T., and Howard, A., "The player/stage project: Tools for multi-robot and distributed sensor systems," in *In Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323, 2003.

[25] Henzinger, T., Preuig, J., and Wong-Toi, H., "Some lessons from the hytech experience," in *Proceedings of the 40th Annual Conference on Decision and Control*, pp. 2887–2892, 2001.

[26] Jill Crisman, C. T., "Scarf: A color vision system that tracks roads and interections," *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 49–58, Feb 1993.

[27] Julius, A. and Pappas, G. J., "Probabilistic testing for stochastic hybrid systems," in *IEEE Conference on Decision and Control*, December 2008.

[28] Julius, A. A., Halasz, A., Sakar, M. S., Rubin, H., V.Kumar, and Pappas, G. J., "Stochastic modeling and control of biological systems: the lactose regulation system of escherichia coli," *IEEE Transactions on Automatic Control*, vol. 53, pp. 51–65, January 2008.

[29] Kim, D., Sun, J., Min, S., James, O., Rehg, M., and Bobick, A. F., "Traversability classification using unsupervised on-line visual learning for outdoor robot navigation," in *In Proc. of Int'l Conf. on Robotics and Automation (ICRA). IEEE*, 2006.

[30] Kluge, K. C., Kreucher, C. M., and Lakshmanan, S., "Tracking lane and pavement edges using deformable templates," in *Enhanced and Synthetic Vision* (Verly, J. G., ed.), vol. 3364, pp. 167–176, SPIE, 1998.

[31] Koenig, S. and Likhachev, M., "D*-lite," in *National Conference on Artificial Intelligence*, pp. 476–483, 2002.

[32] LaValle, S. M., *Planning Algorithms*. Cambridge University Press, 2006.

[33] Lin, L.-J., "Hierarchical learning of robot skills by reinforcement," in *International Conference on Neural Networks*, 1993.

[34] Lygeros, J., Tomlin, C., and Sastry, S., "Controllers for reachability specifications for hybrid systems," *Automatica*, vol. 35, pp. 349–370, 1999.

[35] Mitchell, T., *Machine Learning*. McGraw-Hill, 1997.

[36] Nilsson, N., "Shakey the robot," Technical Note 323, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1984.

[37] Pappas, G., Lafferriere, G., and Shastry, S., "Hierarchically consistent control systems," *IEEE Transactions on Automatic Control*, vol. 45, pp. 1144–1160, 2000.

[38] Parker, L. E., "Distributed algorithms for multi-robot observation of multiple moving targets," *Autonomous Robots*, vol. 12, pp. 231–255, 2002.

[39] Powers, M. and Balch, T., "An incremental approach to adaptive integration of layers of a hybrid control architecture," in *International Workshop on Hybrid Control of Autonomous Systems*, 2009.

[40] Powers, M. and Balch, T., "A learning approach to integration of layers of a hybrid control architecture," in *IEEE International Conference Intelligent Robots and Systems*, 2009.

[41] Rosenblatt, J., "Damn: A distributed architecture for mobile navigation," in *Journal of Experimental and Theoretical Artificial Intelligence*, pp. 339–360, AAAI Press, 1997.

[42] Rosenblatt, J., "Optimal selection of uncertain actions by maximizing expected utility," *Autonomous Robots*, vol. 9, no. 1, pp. 17–25, 2000.

[43] Stone, P., *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1998.

[44] Sukthankar, R., Pomerleau, D., and Thorpe, C., "A distributed tactical reasoning framework for intelligent vehicles," *SPIE: Intelligent Systems and Manufacturing*, 1997.

[45] Sun, J., Mehta, T., Wooden, D., Powers, M., Rehg, J., Balch, T., and Egerstedt, M., "Learning from examples in unstructured, outdoor environments," *Journal of Field Robotics*, vol. 23, pp. 1019–1036, November/December 2006.

[46] Sutton, R. and Barto, A., *Reinforcement Learning, an Introduction*. The MIT Press, 1998.

[47] Thrun, S., "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium* (Lakemeyer, G. and Nebel, B., eds.), Morgan Kaufmann, 2002.

[48] Thrun, S., Burgard, W., and Fox, D., *Probabilistic Robotics*. The MIT Press, 2005.

[49] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohb, S., Dupont, C., erik Jendrossek, L., Koelen, C., Markey, C., Rummel, C., Niekerk, J. V., Jensen, E., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P., "The robot that won the darpa grand challenge," *Journal of Field Robotics*, vol. 23, pp. 661–692, 2006.

[50] Tomlin, C., Lygeros, J., and Sastry, S., "Synthesizing controllers for nonlinear hybrid systems," in *Hybrid Systems: Computation and Control*, pp. 360–373, Springer-Verlag, 1998.

[51] Tomlin, C. J., Lygeros, J., and Sastry, S. S., "A game theoretic approach to controller design for hybrid systems," in *Proceedings of the IEEE*, pp. 949–970, 2000.

[52] Urmson, C., Anhalt, J., Bae, H., Bagnell, J. A., Baker, C., Bittner, R. E., Brown, T., Clark, M. N., Darms, M., Demitrish, D., Dolan, J., Duggins, D., Ferguson, D., Galatali, T., Geyer, C. M., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T., Kolski, S., Likhachev, M., Litkouhi, B., Kelly, A., McNaughton, M., Miller, N., Nickolaou, J., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Sadekar, V., Salesky, B., Seo, Y.-W., Singh, S., Snider, J. M., Struble, J. C., Stentz, A. T., Taylor, M., Whittaker, W. R. L., Wolkowicki, Z., Zhang, W., and Ziglar, J., "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, vol. 25, pp. 425–466, June 2008.

[53] Williams, S., Newman, P., Rosenblatt, J., Dissanayake, G., and Durrant-Whyte, H., "Autonomous underwater navigation and control," *Robotica*, vol. 19, no. 5, pp. 481–496, 2001.

[54] Wooden, D., "A guide to vision-based mapping," *IEEE Robotics & Automation Magazine*, vol. 13, pp. 94–98, June 2006.

[55] Wooden, D. and Egerstedt, M., "Oriented visibility graphs: Low-complexity planning in real-time environments," in *IEEE International Conference on Robotics and Automation*, June 2006.

[56] Wooden, D., Powers, M., Egerstedt, M., Christensen, H., and Balch, T., "A modular, hybrid system architecture for autonomous, urban driving," *Journal of Aerospace Computing, Information, and Communication*, vol. 4, pp. 1047–1058, December 2007.

[57] Wooden, D., Powers, M., MacKenzie, D. C., Balch, T., and Egerstedt, M., "Control-driven mapping and planning," in *IEEE International Conference Intelligent Robots and Systems*, November 2007.

[58] Young, H. and Freedman, R., *Sears and Zemansky's University Physics*, vol. 2. Addison-Wesley, 2000.